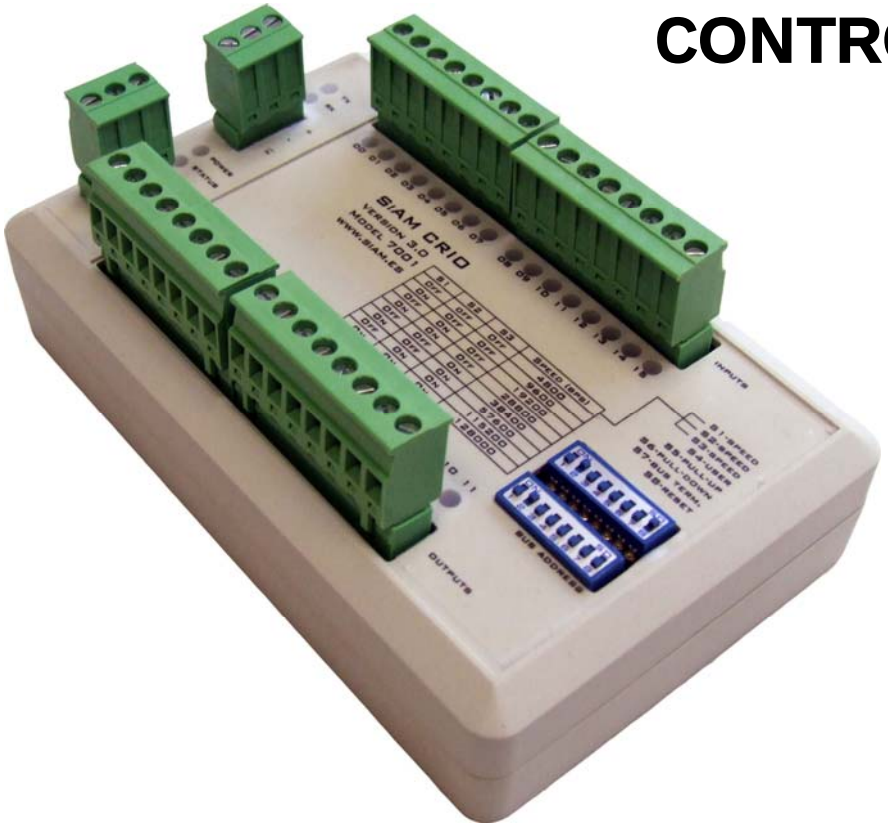
SiAM CRIO v3.0

PROGRAMMABLE REMOTE INPUT/OUTPUT CONTROLLER



User Manual

Version 3.0

CONTENTS

1.- CHARACTERISTICS 4
2.- INTRODUCTION 5
3.- INSTALLATION 8
4.- WIRING 9
4.1.- Main power supply 9
4.2.- Communication 10
4.2.1.- RS-485 module 10
4.2.2.- RS-232 module 11
4.3.- Digital outputs 12
4.4.- Digital inputs 14
4.5.- Analog inputs 15
4.6.- Wiring example 18
5.- BOARD SETTINGS 19
5.1.- Bus address 19
5.2.- Baud rate 20
5.3.- User-defined switch 20
5.4.- Communication resistors 21
5.5.- Reset switch 21
6.- SOFTWARE 22
6.1.- Online mode 23
6.2.- Inputs and outputs 24
6.3.- RAM memory (M) 25
6.4.- EEPROM memory (V) 25
6.5.- System memory 25
6.6.- Virtual registers 26
6.7.- Status 27
6.8.- Configuration 27
6.8.1.- Communication and program 28
6.8.2.- Persistent virtual registers 29
6.8.3.- Analog inputs scaling 31
6.9.- Program 32
6.10.- Other device options 35
7.- IEC PROGRAMMING 36
7.1.- The instruction list (IL) languange 36
7.2.- Data types 37
7.3.- Instructions 37
7.4.- Memory areas 43
7.4.1.- Inputs memory 44
7.4.2.- Outputs memory 44
7.4.3.- RAM memory 44
7.4.4.- EEPROM memory 44
7.4.5.- System memory 45
7.5.- Program errors and warnings 51
7.6.- Programming examples 53
7.6.1.- Simple automatic door 53

7.6.2.- Automatic door with local and remote control 54
7.6.3.- Timers 55
7.6.4.- Intermitencies 56
7.6.5.- EEPROM checksum management 58
7.6.6.- Use of indirect addressing registers 60
8.- MODBUS COMMUNICATION 61
8.1.- Modbus protocol 61
8.2.- Modbus mapping 62
8.2.1.- Indirect addressing registers 63
8.2.2.- Configuration registers 63
8.3.- Modbus functions 64
8.3.1.- Read Holding Registers (0x03) 64
8.3.2.- Read Input Registers (0x04) 64
8.3.3.- Write Single Register (0x06) 65
8.3.4.- Write Multple Registers (0x10) 65
8.3.5.- Mask Write Register (0x16) 66
8.3.6.- Read/Write Multiple Registers (0x17) 67
9.- FREQUENTLY ASKED QUESTIONS 68
9.1.- Communication with the board can not be established 68
9.2.- Digital outputs are all inactive 68
9.3.- The IEC code is not executing correctly 69
10.- ELECTRICAL CHARACTERISTICS 70
11.- COPYRIGHT 71

1.- CHARACTERISTICS

1. Programmable controller with integrated inputs/outputs:
    - Programmable in instruction list (IL) language, according to IEC 1131-3 standard.
    - 8192 program bytes (1300-8000 IEC instructions).
    - 512 user words (256 retentive, 256 non-retentive).
    - Supports the following IEC data types: BOOL, BYTE, SINT, WORD, INT, DWORD, DINT, REAL.
2. Inputs/ouputs:
    - 16 optoisolated 12/24VDC digital inputs.
    - 12 optoisolated 12/24VDC digital outputs.
    - Up to eight 0-5V or 0-20mA analog inputs (10-bit resolution).
    - Indepentent supply for digital outputs.
3. Communications:
    - RS-485 (up to 64 devices in the bus) or RS-232 serial communication.
    - Operation at 8 different speeds: 4800 bps, 9600 bps, 19200 bps, 28800 bps, 38400 bps, 57600 bps, 115200 bps, 128000 bps.
    - Bus termination resistor to avoid reflections, and pull-up and pull-down resistors to stabilize bus (jumper selectable).
    - LED indication for state of reception and transmisssion.
    - Modbus RTU slave protocol. The following commands are supported:
        - 03 (0x03) Read Holding Registers
        - 04 (0x04) Read Input Registers
        - 06 (0x06) Write Single Register
        - 16 (0x10) Write Multiple Registers
        - 17 (0x11) Report Slave ID
        - 22 (0x16) Mask Write Register
        - 23 (0x17) Read/Write Multiple Registers
        - 43/14 (0x2B/0x0D) Read Device Identification
    - Indirect addressing modbus registers for optimizing communication cycles.
4. External power supply 9-30VDC.
5. LED indication for the board power supply.
6. Free software for programming, configuration and diagnostics.
7. Plug-in screw connectors.
8. DIN-rail mounting enclosure.
9. Size: 144 mm x 90 mm x 31mm.

2.- INTRODUCTION

SiAM CRIO is a programmable remote input/output controller with integrated modbus slave communication. It can be used as a remote I/O module in a fieldbus controlled by any device that implements the modbus master protocol, such as computers (PCs), programmable controllers (PLCs), operator panels (HMIs), etc.

This fieldbus topology has great advantages over centralised control systems, because allows the distribution of I/O modules reducing the length of the wiring for sensors and actuators, and reducing also the size of the electric cabinets. Besides, the central controller (PC or PLC) can be placed anywhere in the plant, in the control room for example.

This new version can also work as a stand-alone programmable logic controller or PLC, executing small control tasks that can be programmed in the instruction-list (IL) language defined in the IEC 1131-3 international standard. Both operating modes –remote I/O module and PLC- can be combined to perform a descentralised control.

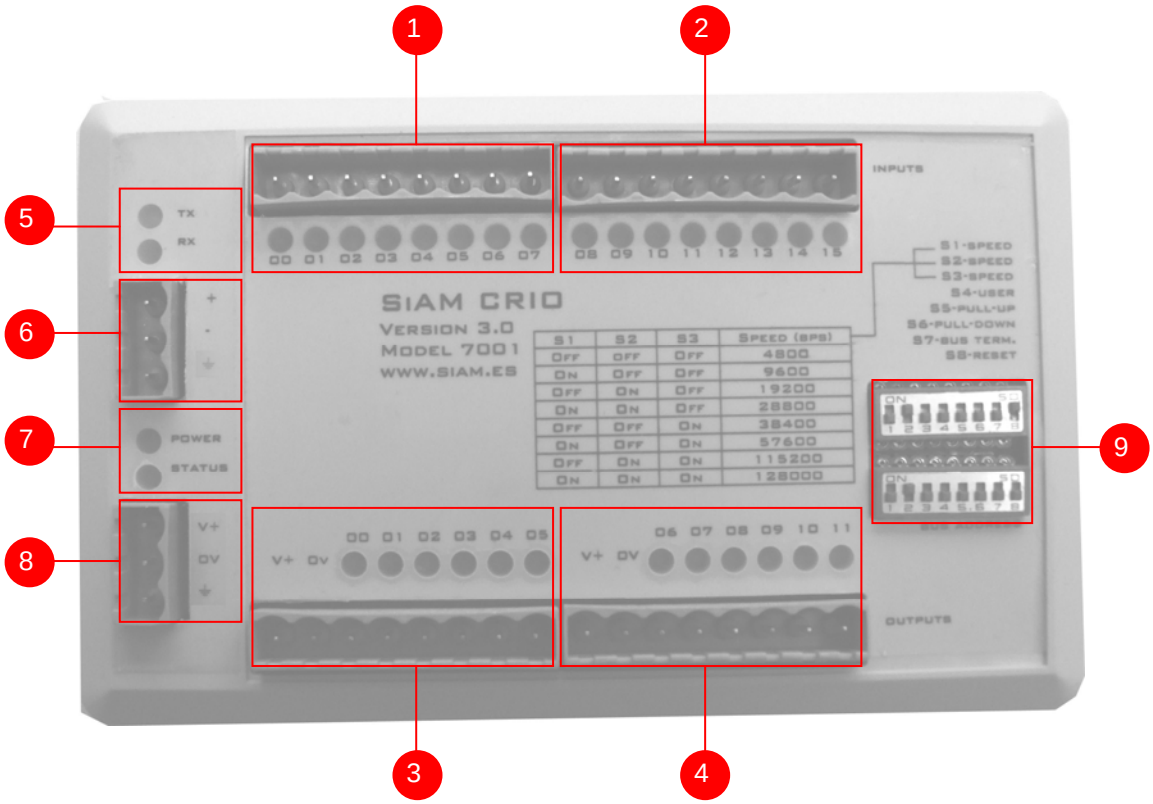
The SiAM CRIO module has 12 digital outputs (DO), 16 digital inputs (DI) and 8 analog inputs (AI), which are multiplexed with digital inputs 8-15.

Up to 255 SIAM CRIO modules can be connected in a modbus network, although this protocol only allows addressing of 247 nodes. So, in a modbus network we could have a maximum of 2964 digital outputs, 3952 digital inputs and 1976 analog inputs.

There are two available versions of the module: one with RS-485 port (3-way screw connector) and another one with RS-232 port (DB9 male connector).

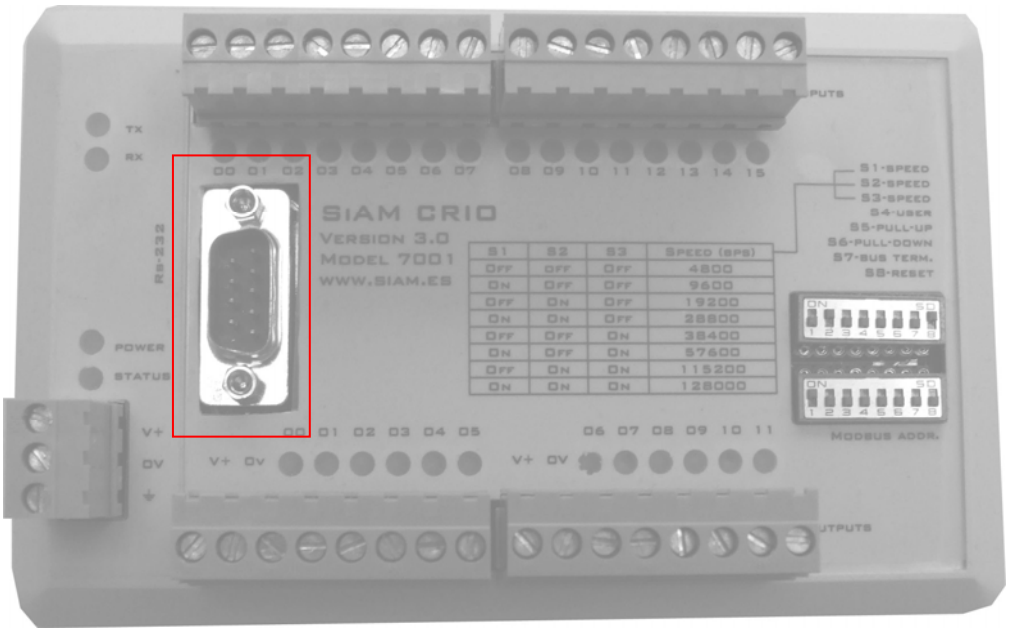
The configuration, programming and monitoring of the module can be done with the SiAM CRIOSoft software, which can be freely downloaded from **www.siam.es**.

This picture displays the RS-485 module version, with its LEDs and connectors:



1. Connector and LEDs for digital inputs DI00…DI07
2. Connector and LEDs for digital inputs DI08…DI15 and analog inputs AI0…AI7
3. Connector and LEDs for digital outputs DO00…DO05
4. Connector and LEDs for digital outputs DO06…DO11
5. Communication bus status LEDs
6. Communication bus connector (RS-485)
7. Power and module status LEDs
8. Power connector
9. Configuration switches

The RS-232 module version only differs in the communication connector, as can be seen in this picture:

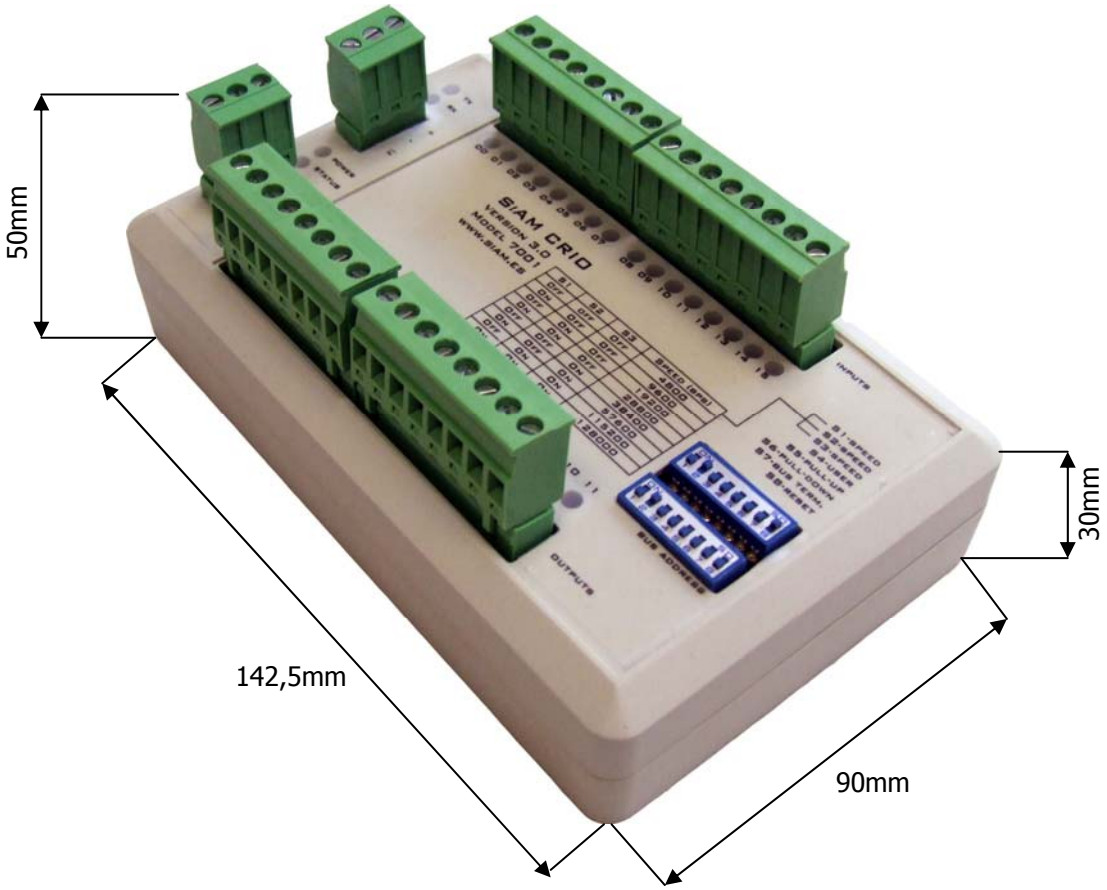


The rest of the connectors and LEDs are the same in both versions.

3.- INSTALLATION

The SiAM CRIO board, as any electronic controller, must be placed as far as possible from heat sources, power lines and any source of electrical noise such as relays or AC motor drives.

The dimensions of the board are these:

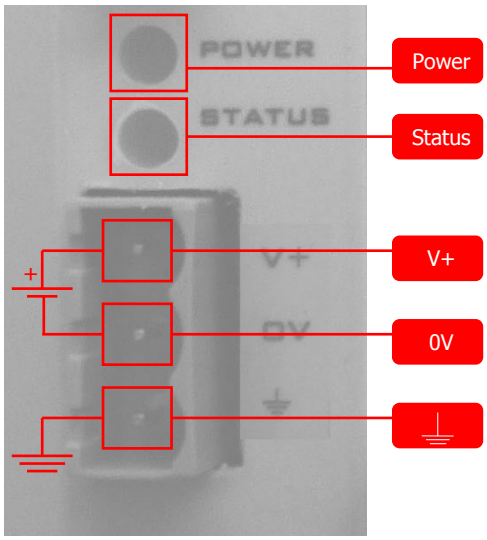


The board enclosure is equipped with two DIN-rail mounting clips, which make installation very easy. To remove the board from the DIN-rail, pull with a flat screwdriver to release the clips.

4.- WIRING

4.1.- Main power supply

Power supply voltage supplies power to the electronic devices in the board. It can be any voltage between 9 and 30 VDC, and is connected through the connector shown in the following figure:



Connector	Description
V+	Supply voltage (9-30 VDC)
0V	Supply ground
⏚	Earth

The Power LED will turn on to indicate that the board is powered. If the board is powered but this led is not lit, then there is a hardware problem and the board must be repaired.

The Status LED indicates the current operating state of the device, according to the following table:

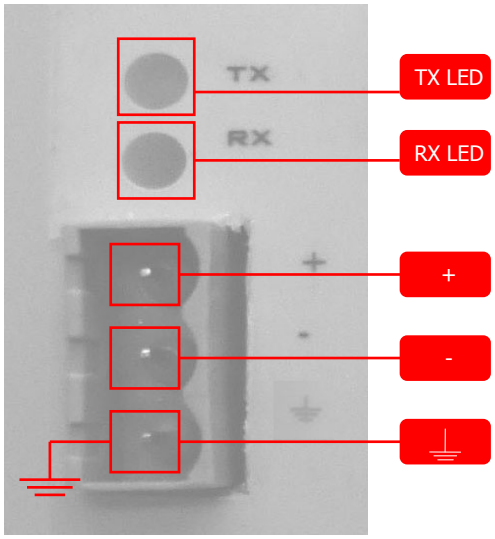
Blink type	Device status
1 second on, 1 second off	STOP mode. Program is empty or stopped by the user.
1 fast blink (0.1 seconds on, 0.5 seconds off)	RUN mode. Program is executing correctly with no error.
2 fast blinks	RUN mode. Program is executing with warnings. Use the programming software to determine the warning condition.
3 fast blinks	STOP mode. Program execution was aborted because of programming errors. Use the programming software to determine the error condition.

This LED must be always blinking. If it's always on or always off, then there is a hardware problem and the board must be repaired.

4.2.- Communication

4.2.1.- RS-485 module

This version uses a 3-way screw connector described in the following figure to implement a 2-wire bus:



Connector	Description
+	RS-485 positive terminal, also known as 485+, A line or modbus D0 line
-	RS-485 negative terminal, also known as 485-, B line or modbus D1 line
⏚	Cable shield. It's internally connected to power supply ground.

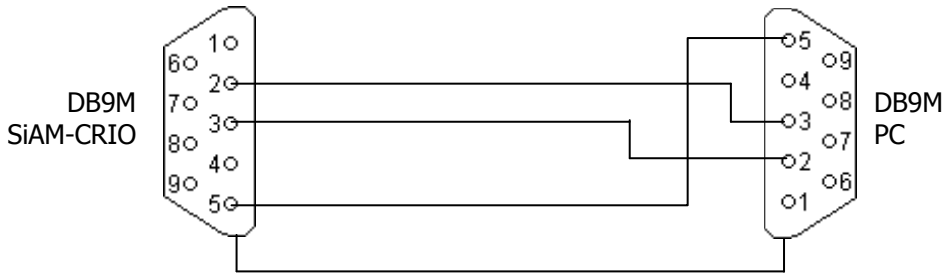
TX and RX LEDs will indicate the status of the communications. The TX LED will turn on when the device is transmitting data to the RS-485 bus.

The RX LED will turn on when data is received from the RS-485 bus. Only valid serial data detected in the bus will turn this LED on. If there is traffic in the bus at a different baud rate that the one configured in the slave, this LED will stay off.

As in any RS-485 network, bus termination is required at both ends of the bus. Also, line polarization can be necessary for some devices. All these can be done with the configuration switches described in section 5.

4.2.2.- RS-232 module

This version uses a standard DB9 male connector, which can be connected to a PC serial port with a normal null-modem cable. The pinout of this cable is shown in the following diagram:

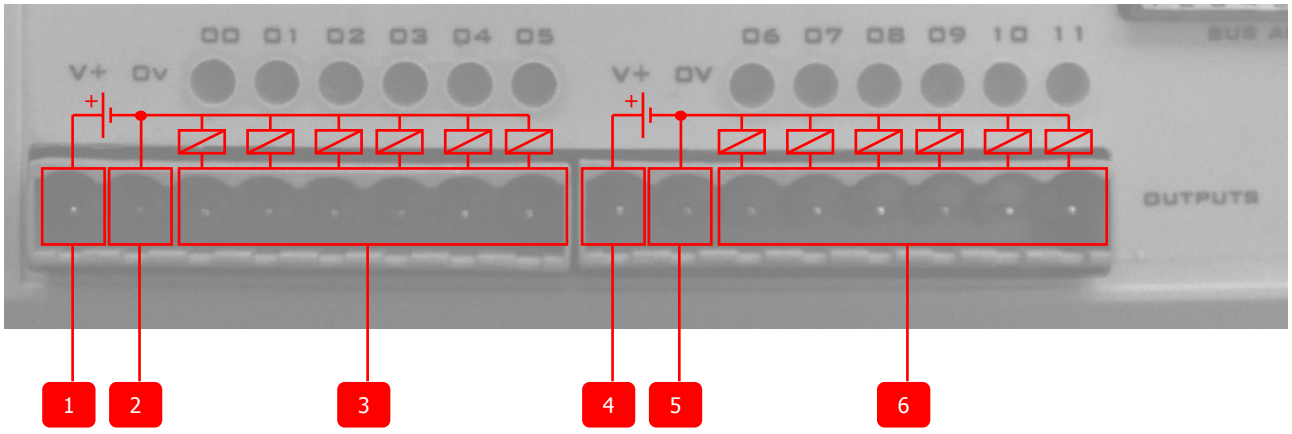


Connecting DB9 shields is optional though recommended to increase noise immunity. Connection to serial ports with DB25 connectors is also possible using standard DB25-DB9 adapters.

TX and RX LEDs will indicate the status of the communications, just like in the RS-485 version.

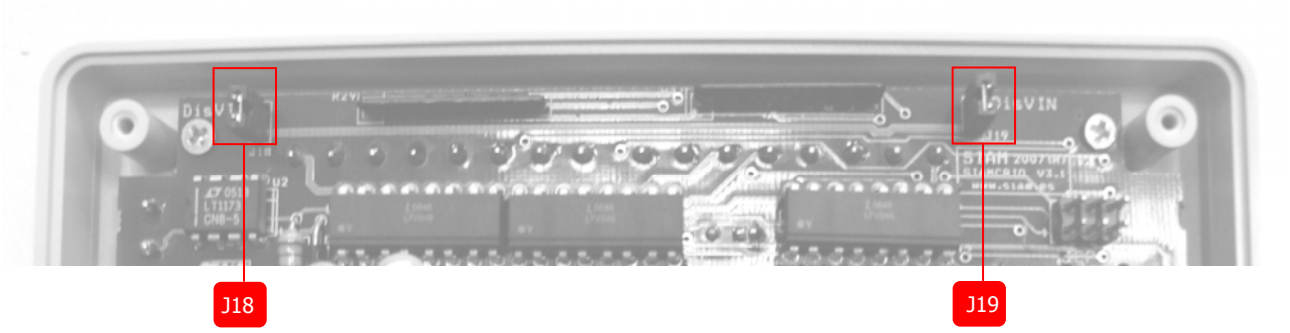
4.3.- Digital outputs

The board has 12 optoisolated PNP-Darlington digital outputs, with independent supply for outputs 0…5 and outputs 6…11. The connection of the outputs is done in the following connectors:



Connector	Description	Comments
1	Power supply for digital outputs DO00-DO05 (9VDC…30VDC)	Connect here the positive supply voltage for the first group of digital outputs. This independent supply can be interrupted –with an emergency stop or a safety relay for example- to deactivate this group of outputs without affecting the others.
2	Power supply for digital outputs DO00-DO05 (GND)	This is the power supply ground for the first group of digital outputs. It's internally connected to the main power supply ground.
3	Connectors for digital outputs DO00-DO05	Connect here digital outputs DO00…DO05. When the corresponding output is active, its LED will be on and its terminal will have the voltage supplied in connector 1 (9…30VDC, typically 24VDC). Otherwise, voltage in the terminal will be 0VDC and its LED will be off.
4	Power supply for digital outputs DO06-DO11 (9VDC…30VDC)	Connect here the positive supply voltage for the second group of digital outputs. This independent supply can be interrupted –with an emergency stop or a safety relay for example- to deactivate this group of outputs without affecting the others.
5	Power supply for digital outputs DO06-DO11 (GND)	This is the power supply ground for the second group of digital outputs. It's internally connected to the main power supply ground.
6	Connectors for digital outputs DO06-DO11	Connect here digital outputs DO06…DO11. When the corresponding output is active, its LED will be on and its terminal will have the voltage supplied in connector 4 (9…30VDC, typically 24VDC). Otherwise, voltage in the terminal will be 0VDC and its LED will be off.

The use of independent power supplies for the electronics and the outputs allows the use of different voltages, for example 24VDC for the electronics and 12VDC for the outputs. In many cases, though, the same 24VDC power supply is used for both the electronics and the digital outputs. For these cases, the board can bypass the main power supply (positive terminal) to the output terminals. This can be done with two internal jumpers, shown in this figure:



Jumper	Description
J18	Bypass of main positive power supply to positive power supply for outputs DO00…DO05
J19	Bypass of main positive power supply to positive power supply for outputs DO06…DO011

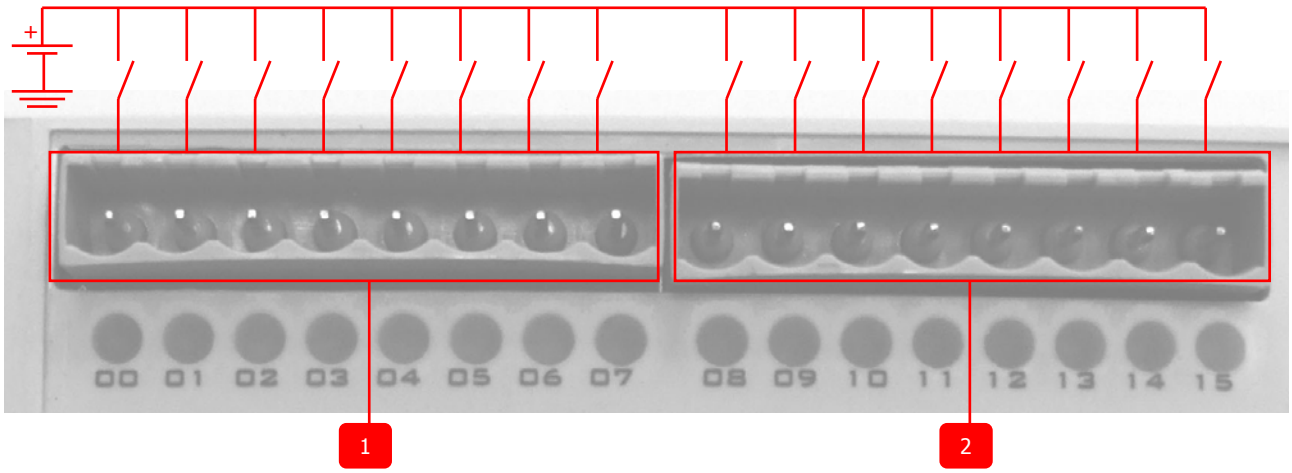
If any of the bypass jumpers is closed, don't connect the power supply terminals in digital output connectors (connectors 1,2,4,5 in previous page) to an independent power supply. Doing so will short-circuit the main power supply with the outputs power supply.

These jumpers bypass only the main power supply positive voltage. The outputs power supply ground is always internally connected to the main power supply ground.

Digital outputs can be activated internally by the IEC user program (%QW0) or externally by modbus communication, by writing a 16-bit register whose first 12 bits are mapped to the 12 digital outputs. An active bit (1) in this register activates its corresponding output, and an inactive bit (0) deactivates the output. If both modbus communication and the user program are controlling the outputs, normally the user program prevails due to its faster execution cycle.

4.4.- Digital inputs

The board has 16 optoisolated digital inputs. The connection of the inputs is done in the following connectors:



Connector	Description	Comments
1	Connectors for digital inputs DI00-DI07	Connect here digital inputs DI00…DO07. When the corresponding input is active (when the voltage in the terminal is 9VDC or higher), its LED will be on. Otherwise, its LED will be off.
2	Connectors for digital inputs DI08-DI15	Connect here digital inputs DI08…DO15. When the corresponding input is active (when the voltage in the terminal is 9VDC or higher), its LED will be on. Otherwise, its LED will be off. These inputs are multiplexed with analog inputs 0…7, which are also connected to this connector. **These group of inputs must be configured by hardware jumpers**, as described in the next section.

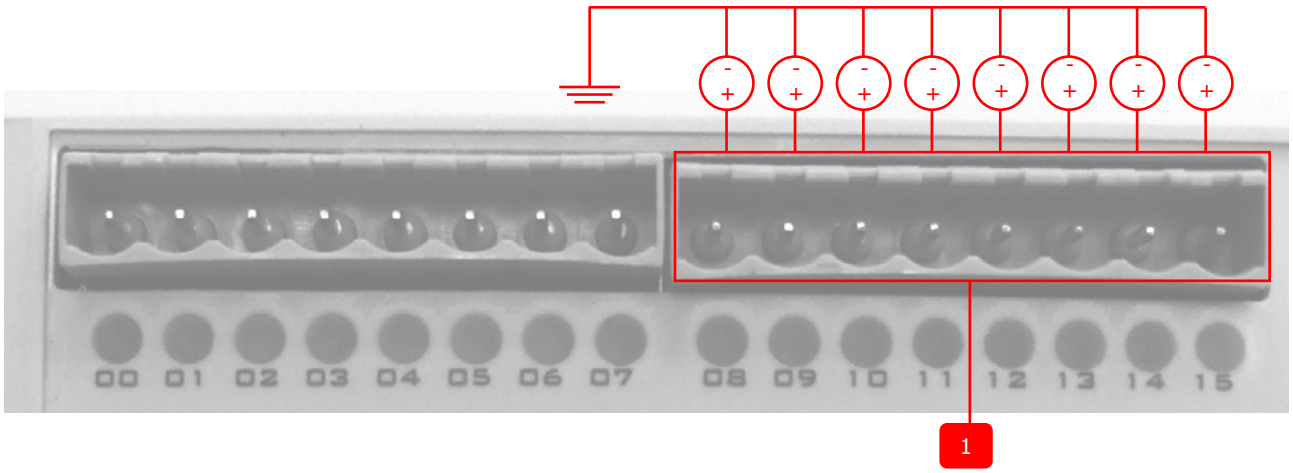
Input voltage for digital inputs must be between 9 and 30VDC, referenced to the main power supply ground.

Digital inputs can be accessed internally by the IEC user program (%IW0) or externally by modbus communication, by reading a 16-bit modbus register whose bits are mapped to the 16 digital inputs. An active bit (1) in this register indicates that its corresponding input is active, and an inactive bit (0) indicates that the input is not active.

Analog inputs are also mapped to this digital inputs register by comparing the analog value of the inputs. So, if an analog input is higher than 2.5V or 10mA, then the corresponding bit in the digital input register will be active.

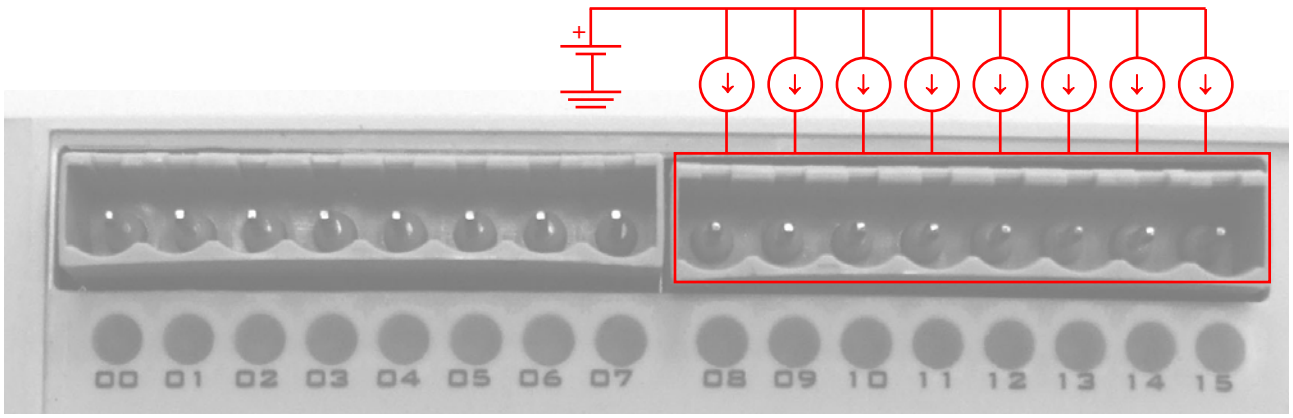
4.5.- Analog inputs

Analog inputs share the same connector for digital inputs DI08…DI15. They can be 0…5VDC voltage inputs -referenced to the main power supply ground- as shown here:



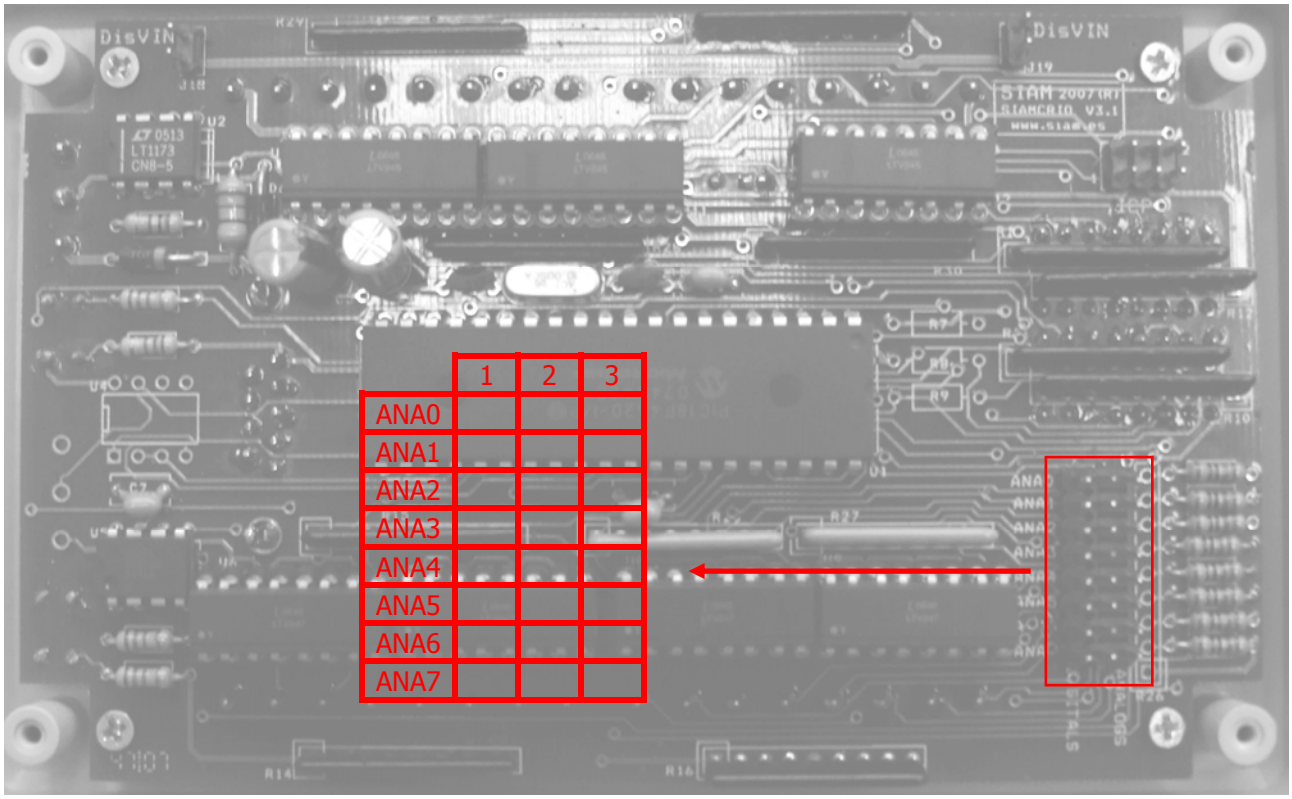
Connector	Description	Comments
1	Connectors for analog inputs AI00-AI07	Connect here analog inputs AI00…AI07.

They can also be 0…20mA current inputs, as shown here:



In this case, analog inputs are passive current inputs. So the current transmitters will not be powered by the analog inputs, and must be powered independently.

Analog inputs must be configured by setting hardware jumpers inside the board. There are 8 jumpers in the board, one for each possible analog input, displayed in the following picture:



To configure any input as analog, close jumpers 1-2. To configure any input as digital, close jumpers 2-3. Any combination of digital-analog inputs is possible.

Do not connect a digital input voltage (24V) to an analog-configured input. Connecting any analog input to a voltage higher than 5V can cause damage to the board.

On the other hand, connecting an analog voltage (0…30V) to a digitally configured input is not dangerous for the board.

Analog inputs are converted with 10-bit resolution, scaled and mapped to 16-bit registers accessible by both the IEC user program (%IW1…%IW8) and modbus communication. These inputs are also thresholded and mapped to the digital input register (bits 7…15).

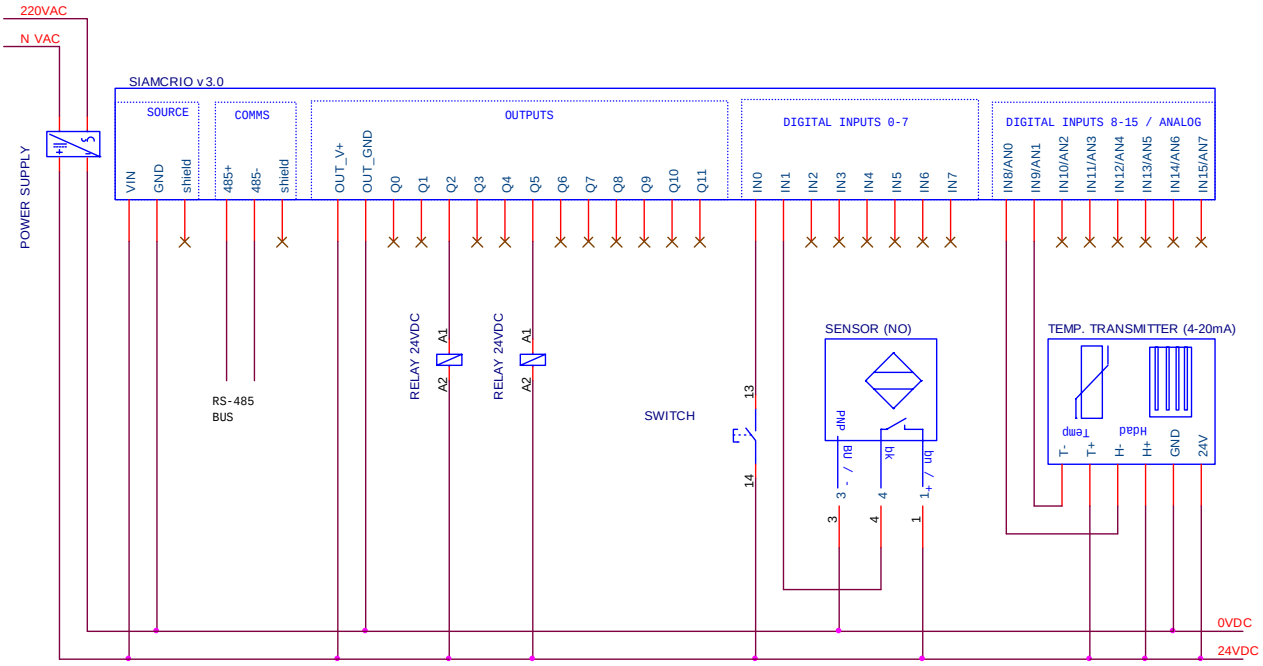
The scaling to the 16-bit registers is performed by a linear conversion whose range can be configured with the programming software. By default, the predefined range is 0-10000. So, 0V/0mA will be converted to a 0 value, and 5V/20mA will be converted to 10000. But this range can be changed to any signed word value between -32768 and +32767.

The unscaled 10-bit conversion value (0…1023) can also be accessed in system registers %SW30…%SW37.

4.6.- Wiring example

The following diagram displays an example of the connection of the board:

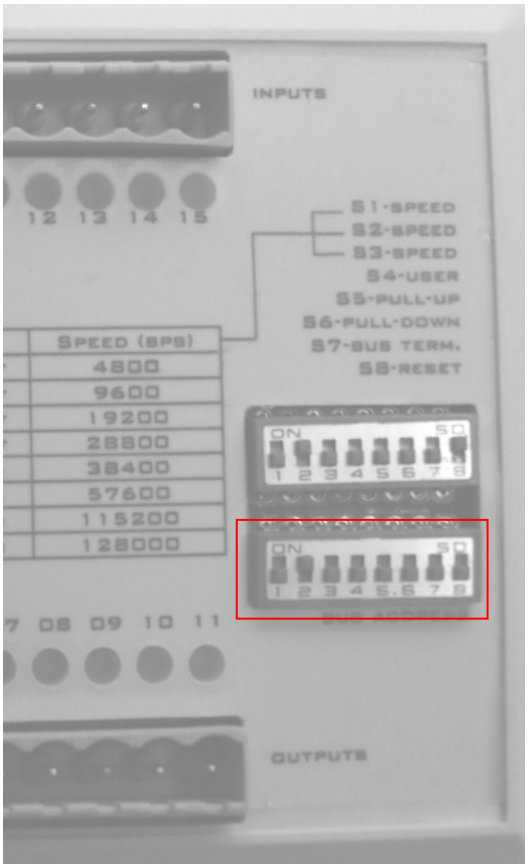
- Outputs %Q0.2 and %Q0.5 are commanding two 24VDC relays.
- Input %I0.0 is connected to a switch.
- Input %I0.1 is connected to a sensor with a normally-open contact.
- Analog inputs %IW0 and %IW0 are reading humidity and temperature from a 4-20mA analog transmitter.



5.- BOARD SETTINGS

5.1.- Bus address

The first step to configure communication is to set up the slave address. This can be done with the lower DIP-switch bank, shown in the picture on the left.



There are 256 selectable addresses, but only addresses between 1 and 247 are valid for the modbus protocol. DIP switch 1 contains the least significant bit of the address, and switch 8 contains the most significant bit.

Set device address 0 to reset the user program. In case of incorrect programming or configuration, power-up the board with all the switches OFF –address 0- and user program will be deleted. In this case, the default board address will be set to 128. The program can then be corrected and downloaded again.

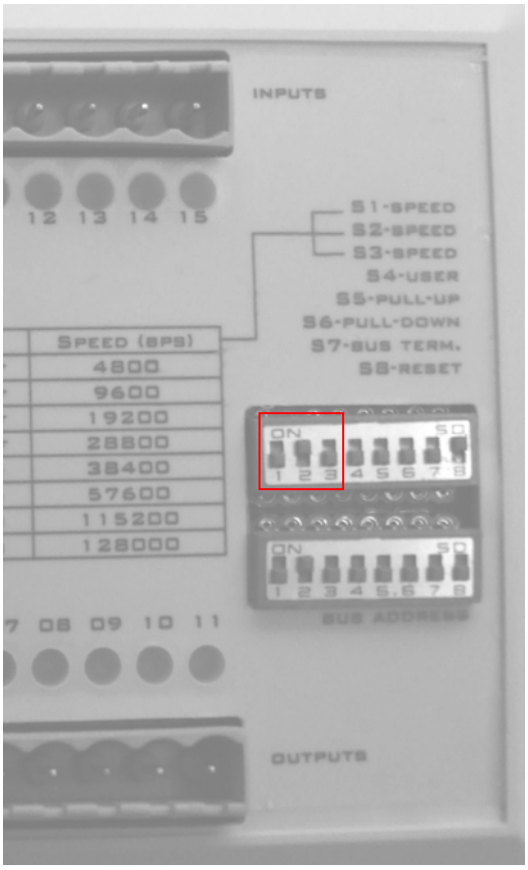
Bus address must be configured for both the RS-232 and RS-485 versions of the module.

On start-up, the bus address bank switch is mapped to system register %SW72 which controls the effective address of the board. This is a read/write register, so the user program can overwrite its value to programmatically change the modbus address.

The bus address switch bank is not read continuously, but only on start-up. So, after changing the modbus address of the board in the switch bank, it must be restarted by removing and reapplying power, or by toggling switch S8 in the upper switch bank as described below.

5.2.- Baud rate

Communication baud rate is selected with switches 1-2-3 from the upper DIP-switch bank, shown in the picture on the left.



The following baud rates can be selected:

S1	S2	S3	Baud rate
OFF	OFF	OFF	4800 bps
OFF	OFF	ON	9600 bps
OFF	ON	OFF	19200 bps
OFF	ON	ON	28800 bps
ON	OFF	OFF	38400 bps
ON	OFF	ON	57600 bps
ON	ON	OFF	115200 bps
ON	ON	ON	128000 bps

As in the previous case, the baud rate selececion is not read continuously, but only on start-up. So, a restart of the board is required after changing this setting.

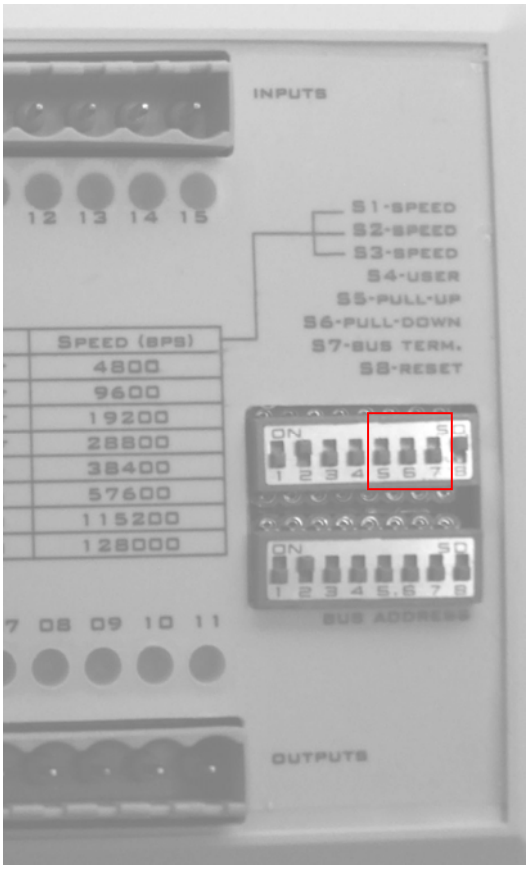
5.3.- User-defined switch

Switch 4 in the upper switch bank has no predefined use. It is mapped on start-up to system bit %S6.11 (all the upper switch bank is mapped to system register %SW6). The IEC user program can check this bit for any user-defined purpose, or just ignore it.

As the rest of the switches, it is only read on start-up. So, a restart of the board is required after changing this switch.

5.4.- Communication resistors

Switches 5-6-7 from the upper DIP-switch bank, shown in the picture on the left, control the communication resistors for RS-485 bus:



Jumper	Description	Comments
S5	**Pull-up** resistor (390 Ω)	Stabilizes the bus. **Must be active in only one node in the RS-485 network.**
S6	**Pull-down** resistor (390 Ω)	Stabilizes the bus. **Must be active in only one node in the RS-485 network.**
S7	**Bus-termination** resistor (220 Ω)	Prevents reflections in the bus. **Must be active only in the terminal nodes in the bus**, and inactive in the all the intermediate nodes.

These switches are hardware switches, so they can be toggled at any time (not only in start-up).

All theses switches are useless in the RS-232 version of the board.

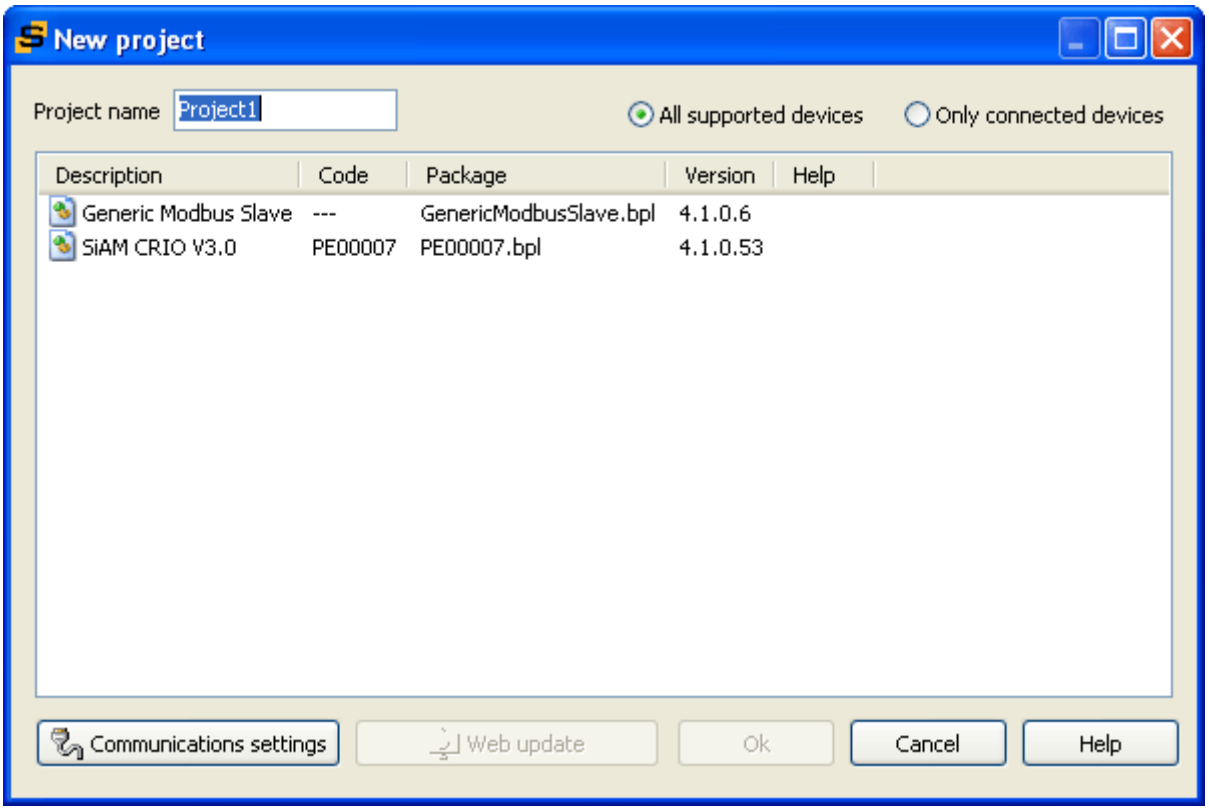
5.5.- Reset switch

Switch 8 in the upper switch bank is used to perform a hardware reset of the board microcontroller, without the need of removing and reapplying power to the whole board. The microcontroller is continuously reset while this switch is OFF.

This can be very useful to quickly update the configuration of the board –modbus address, baud rate- after any change in the configuration switches.

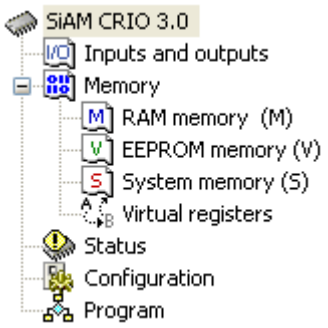
6.- SOFTWARE

The software utility for configuration and programming of the SiAMCRIO board is CRIOSoft, which can be freely downloaded from SiAM web. From CRIOSoft, open a new project and select device **SiAM CRIO v3.0** (PE00007).



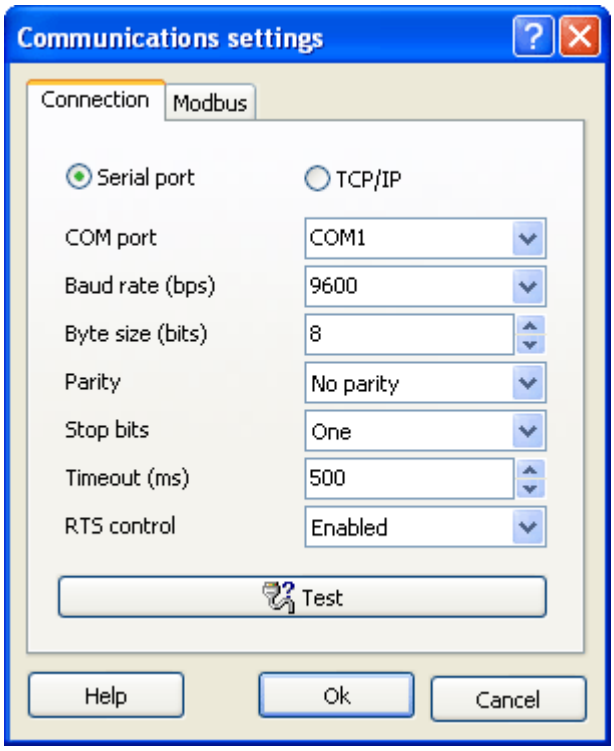
If the board is connected to the modbus network and the serial port is correctly configured, select **Only connected devices** to scan the bus. If the board is found, double-click it to open a new project.

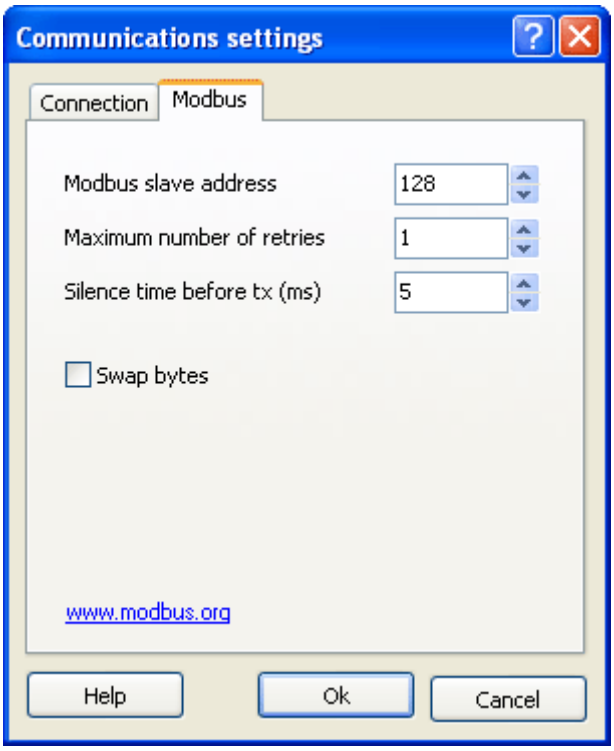
The main device options will show up in the right panel of the CRIOSoft window, as well as some buttons in the top toolbar, and some menus with even more options.



The first thing to do is to set up the **Communication settings** in the **Tools** menu. The settings here have to match those configured with the configuration switches in the board, explained in the previous section (address and baud rate).

The serial format for communications must be **8 data bits**, **1 stop bit** and **no parity**.



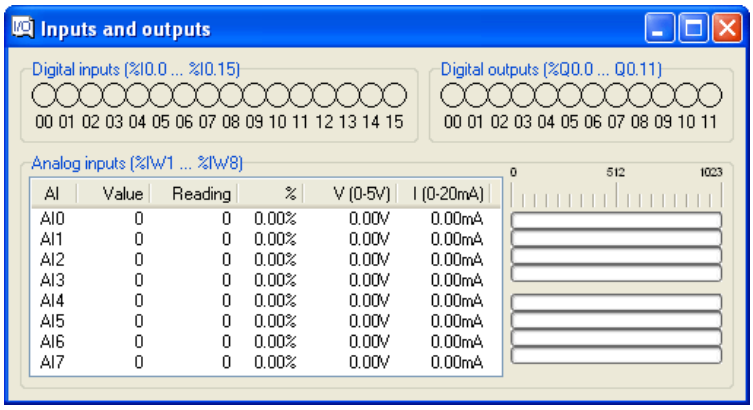


6.1.- Online mode

Monitoring of signals, inputs, outputs, registers, etc… is only possible in **online** mode. To enter online mode, select **Monitor** in **Device** menu, or click the button .

6.2.- Inputs and outputs

The first option displays this window, which displays the status of inputs and outputs in **online** mode.



For any of the analog inputs, you can see:

- The scaled **Value**
- Unscaled **Reading** (0-1023)
- Percentage value (0-100%)
- Voltage and current value.

The state of the digital outputs can be toggled by left-clicking over them.

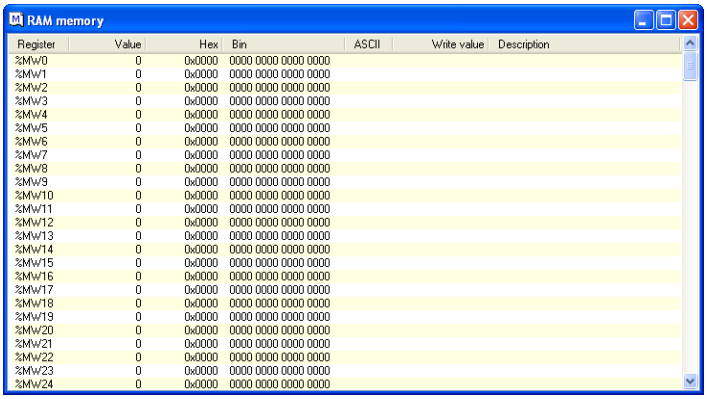
Both digital inputs and outputs can be forced. Any forced input or output will display an **F** over it.

The forcing of an input or output must be done in online mode, by right-clicking over the corresponding indicator. A popup menu will appear, with options to force ON, to force OFF, to delete this force or to delete all forces. Two more options will enable or disable the force masks.

These force masks are volatile, that is, are lost when the board is powered down. Persistent forces of inputs and outputs can be done by means of the IEC user program.

6.3.- RAM memory (M)

The next option displays the volatile registers in the M memory, in decimal, hexadecimal, binary and ASCII formats.



Right-click over any register to display options to change its value, delete it (set to 0) and to assign a textual description.

The write values are stored when in offline mode, and are automatically applied when switching to online mode.

6.4.- EEPROM memory (V)

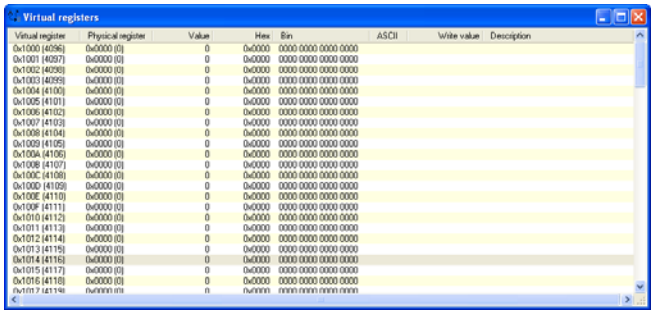
This option displays a window like the previous one, where we can see and modify the values of the non-volatile registers in the V memory of the board.

6.5.- System memory

This option displays a window like the previous one, where we can see (but not modify the values of the system registers in the S memory of the board.

6.6.- Virtual registers

This option displays a window similar to the previous ones, where we can see and modify the virtual or indirect addressing registers of the board.



For any virtual modbus register, we can see the real modbus register it points to, as well as the value of the real register in decimal, hexadecimal, binary and ASCII formats. We can also assign textual descriptions to the virtual registers.

Right-click over a register to change the value of its associated physical register. Besides, we can also change the assignment to real registers, by cliking on the option **Change physical register**.

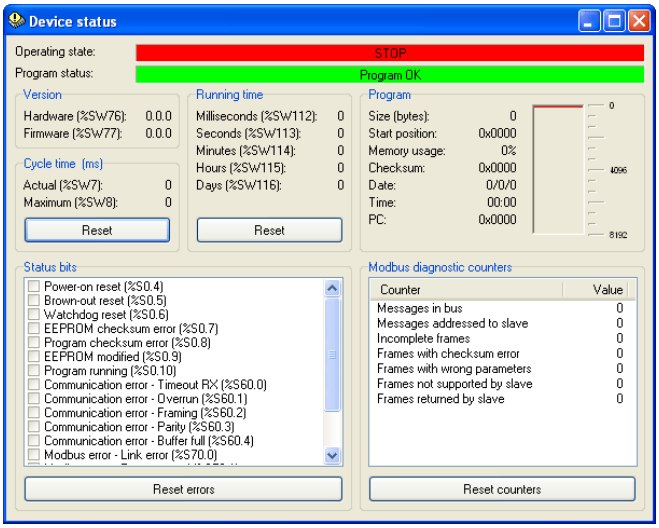
This table can be very useful to efficiently monitor program variables located in scattered or non-contiguous locations in the program, or in different memory areas. That's because all the virtual register area can be read very quickly in one-time, instead of the multiple modbus reads required to read different non-contiguous memory areas. See section 6.8.2 for more information about virtual registers.

These registers are persistent, that is, the association to physical registers is stored in flash memory, so it's not lost on power-down.

6.7.- Status

The status window shows important information about the current status of the board:

- Operating state (RUN or STOP)
- Program status (OK or with error)
- Hardware and firmware version of the board.
- Actual and maximum cycle time of the user program.
- Important status bits.
- Running time counters.
- User program statistics and memory usage bar.



By clicking on the buttons in this window, we can reset the maximum cycle time, the running time and the most important errors.

If there is any error in the user program, and the offline (in CRIOSoft) and online (in the board) programs match, click over the error indicator panel to open the program window and jump to the instruction that generates the error.

6.8.- Configuration

The most important configuration parameters for the board can be set in this window. In the bottom part of the window there are buttons to **Read** and **Write** the current configuration, to accept (**Ok**) or **Cancel** the changes made. These parameters can only be changed in offline mode.

6.8.1.- Communication and program

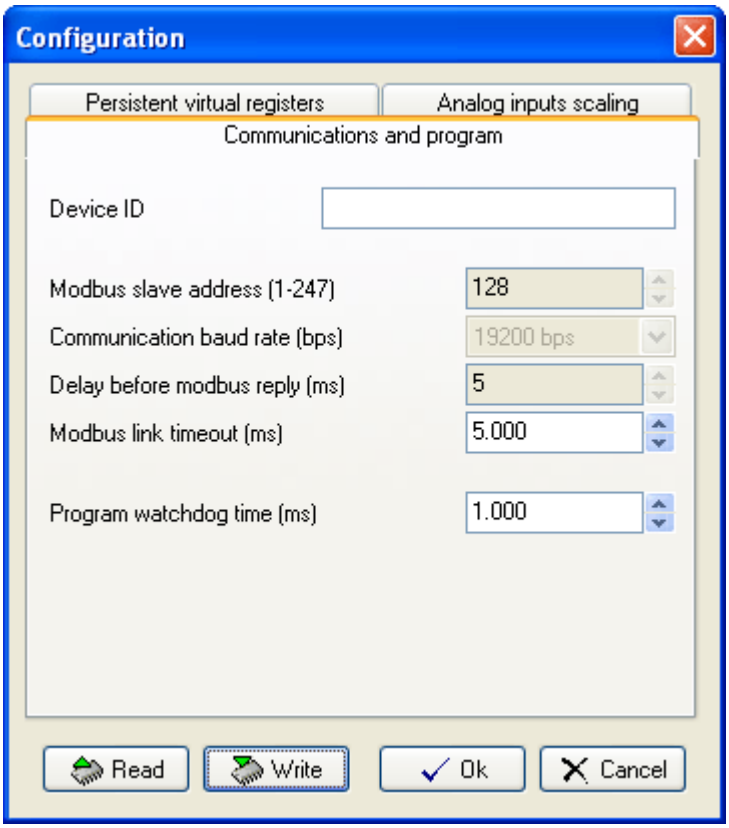
In the first tab, we can set parameters related to communications and the user program:

- **Device ID**. This string (up to 26 characters) textually identifies the modbus slave when scanning the network by means of the **Only connected devices** in the **New Project** option.

- **Modbus slave address**. This value is set via the dip-switch bank, as described in section 5.1

- **Communication baud rate.** This value is set via the dip-switch bank, as described in section 5.2

- **Delay before modbus reply.** This time (in milliseconds) indicates the waiting time befote replying modbus requests from the master. If the master needs a long time to switch transmission direction –for example, in the case of a PC with a RS232/485 converter with software flow control, specially at low speeds (4800bps or lower)- it may be necessary to increase this time.



- **Modbus link timeout**. If this time is exceeded without receiving modbus requests, then the board will consider that the communications link is lost. The state of the communication link is mapped to system bit %S70.0. The user program can use this bit to react in case of loss of communication with the bus master.

- **Program watchdog time**. This is the maximum allowed time for the execution of the complete user program. Exceeding this time causes a fatal error that will stop program execution and turn off all digital outputs.

6.8.2.- Persistent virtual registers

In the second tab we can configure the modbus **indirect addressing**, which is of great use to optimize communication cycle time.

Indirect addressing consists of reading or writing a real or physical modbus register through an indirect or virtual register. This allow the contiguous reading –that is, with only one modbus transaction- of non-contiguous registers scattered across the memory map, which otherwise could only be read with several transactions in a much slower way.

Let's see it with an example, with the following mapping:

- Physical register 0x0000 -> Virtual register 0x1000
- Physical register 0x0100 -> Virtual register 0x1001
- Physical register 0x0200 -> Virtual register 0x1002

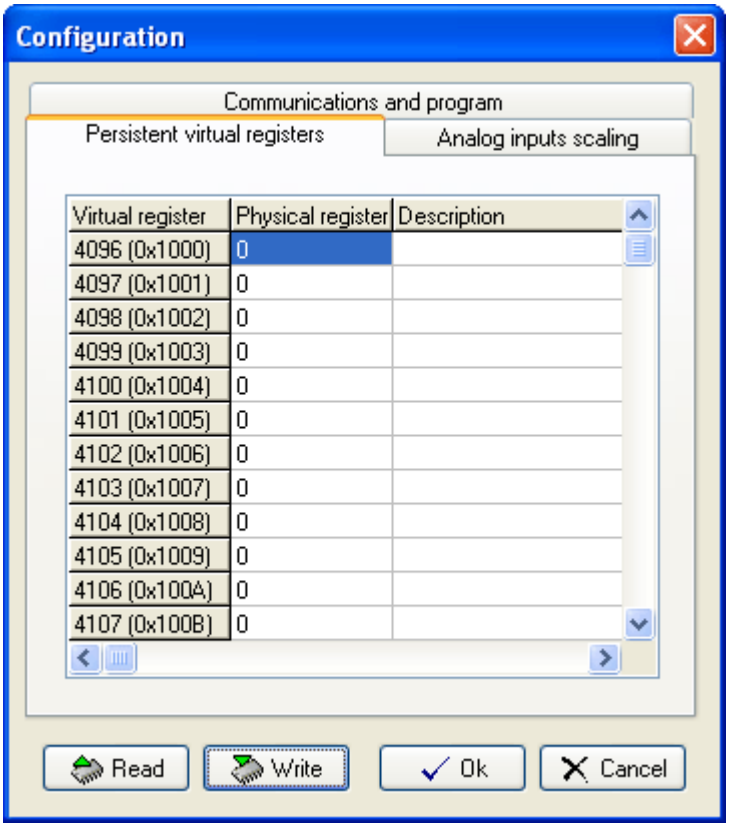
To read physical registers 0x000, 0x0100 and 0x0200, normally we should need 3 modbus transactions (request and reply). But we can also read them by reading the 3 contiguous registers 0x1000, 0x1001 and 0x1002 with only one modbus transaction, for examply by using the frame READ_HOLDING_REGISTERS (code 0x03). We can read and write registers these 3 registers, but we will really be working with the real associated registers. So, writing a value to register 0x1002 will really set the value of register 0x0200.

This mapping or association between real and virtual registers is persistent, so it's not lost on power down.

In CRIOSoft configuration window we can easy configure the 64 available indirect addressing registers (with addresses 0x1000… 0x103F). We can also assign a textual comment to any virtual register.

This assignment table is stored in persistent memory, so this needs to be done only once.

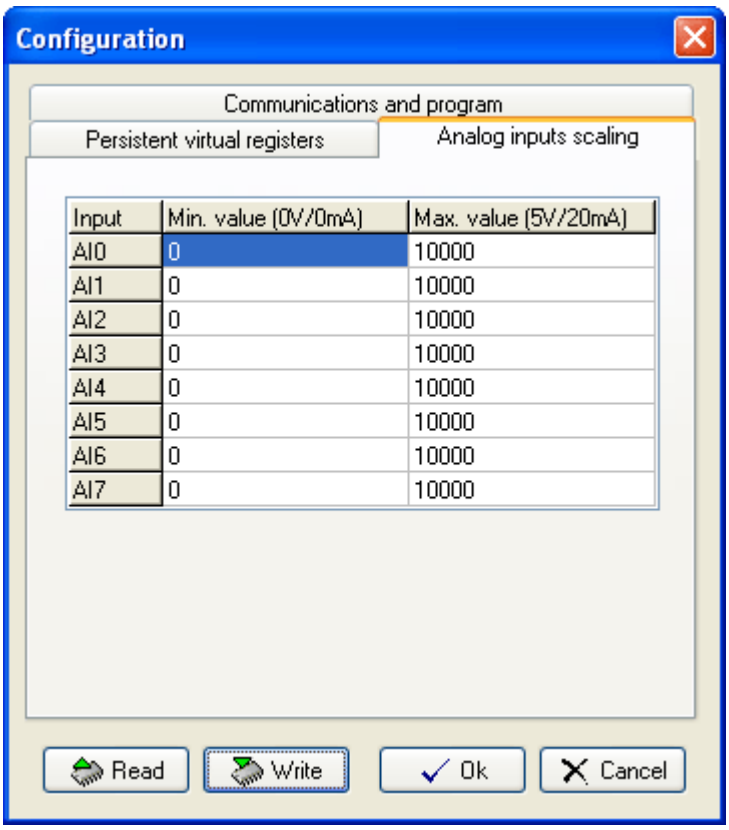
There are another 64 indirect addressing registers (addresses 0x1100…0x113F) which are stored in volatile memory, which allow very easily indirect addressing that must change frequently in time.



6.8.3.- Analog inputs scaling

In the last tab, we can configure the scaling for the analog inputs.

For any input we must specify the minimum and maximum value for the scaling. These values must be in the range -32768… +32767. The maximum value must be higher than the minimum.



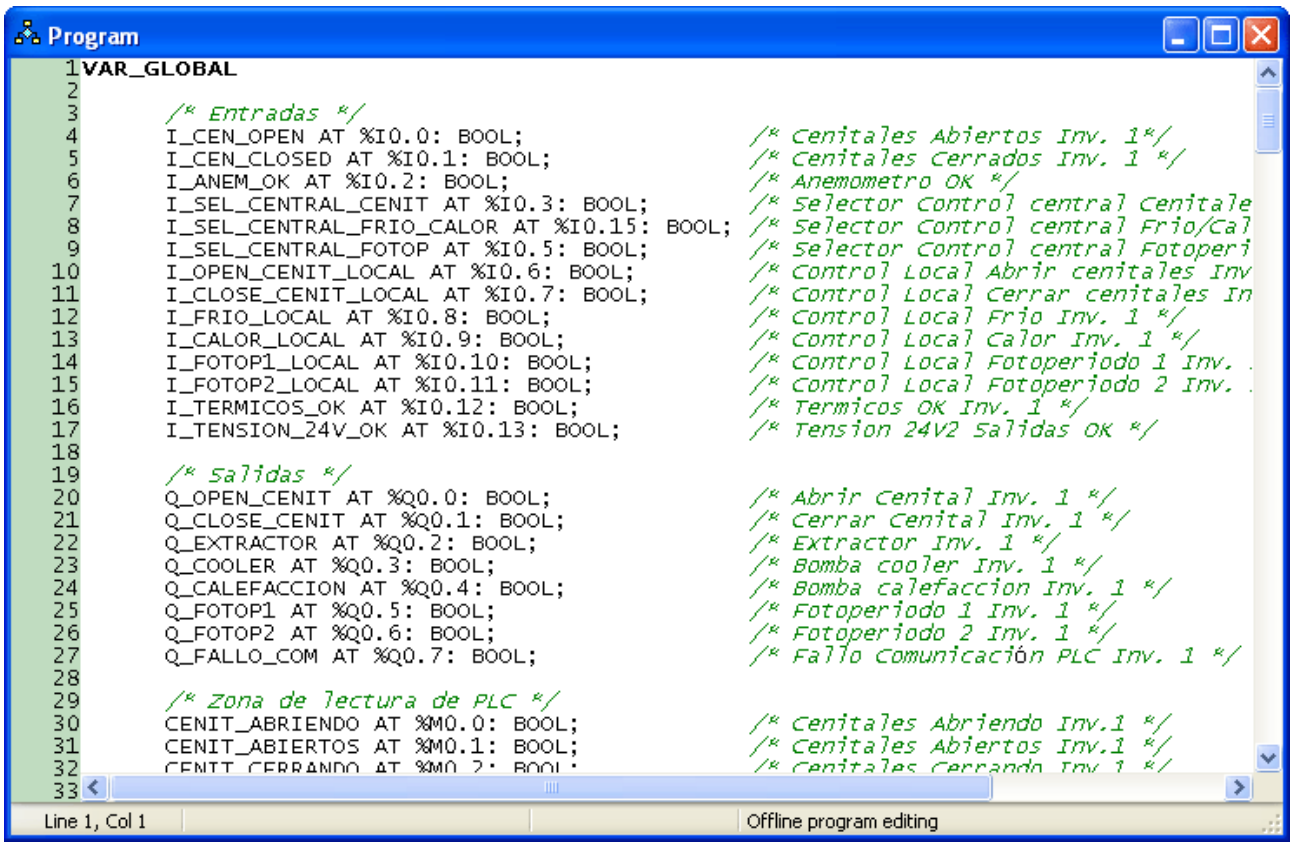
This scaling is very useful in order to work in the user program with engineering units, instead of unscaled 0-1023 values.

For example, if we have a temperature transmitter whose range is with range -40…80ºC and a 0…20mA analog output, we can set the minimum and maximum scaling values to -4000 and 8000. Doing so, in the scaled analog input register (%IW1…%IW8) we would be able to read the real temperature value, multiplied by 100.

Be sure to correctly configure the analog inputs with the internal switches -as described in section 4.5- because connection of a high voltage (24VDC for example) to any input configured as analog can cause damage the board.

6.9.- Program

The program window allows editing of the instruction list user program, and also its monitoring in online mode.

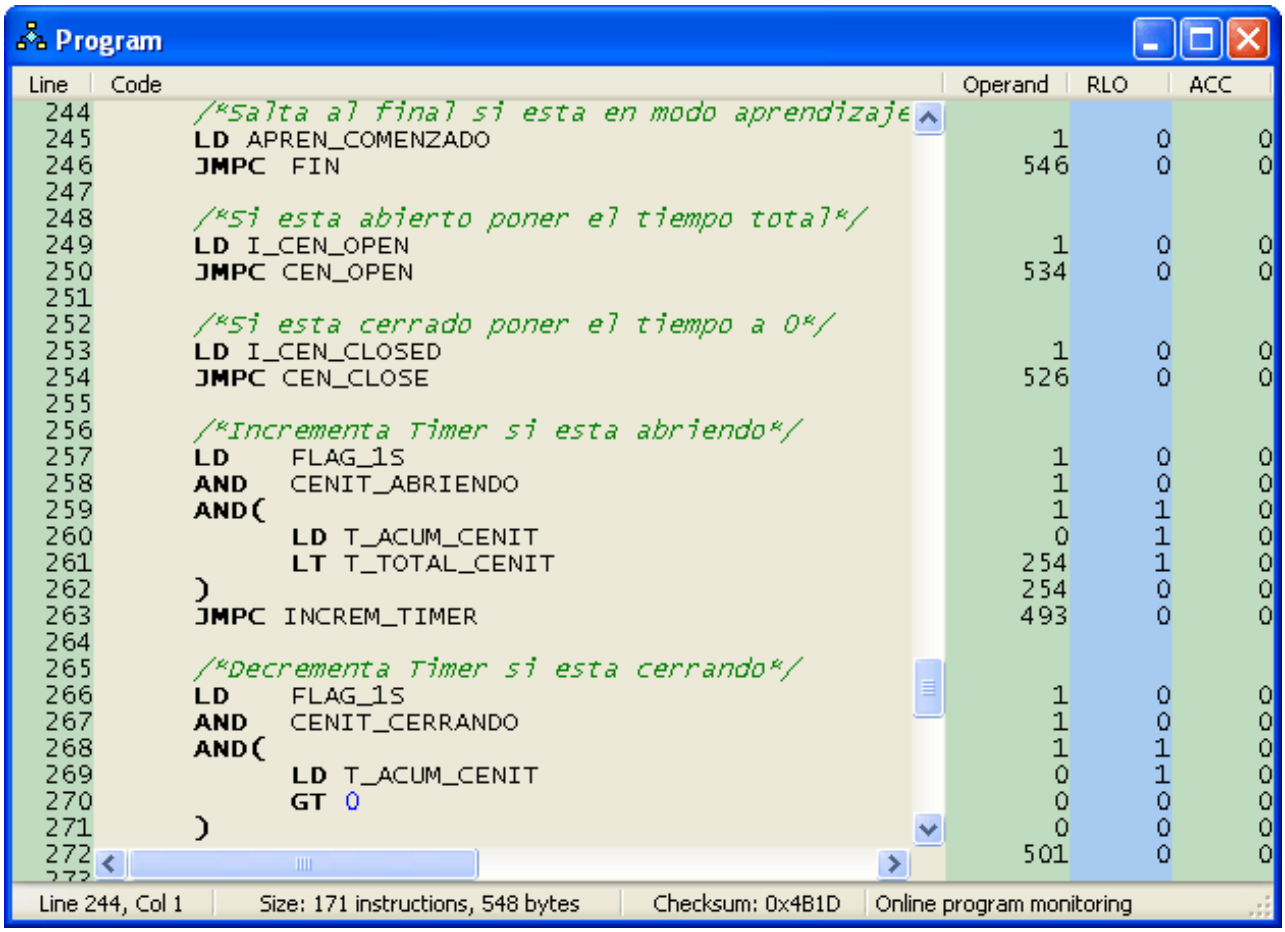

Program
VAR_GLOBAL
/* Entradas */
I_CEN_OPEN AT %I0.0: BOOL; /* Cenitales Abiertos Inv. 1*/
I_CEN_CLOSED AT %I0.1: BOOL; /* Cenitales Cerrados Inv. 1 */
I_ANEM_OK AT %I0.2: BOOL; /* Anemometro OK */
I_SEL_CENTRAL_CENIT AT %I0.3: BOOL; /* Selector Control central Cenitale
I_SEL_CENTRAL_FRIO_CALOR AT %I0.15: BOOL; /* Selector Control central Frio/Cal
I_SEL_CENTRAL_FOTOP AT %I0.5: BOOL; /* Selector Control central Fotoperi
I_OPEN_CENIT_LOCAL AT %I0.6: BOOL; /* Control Local Abrir cenitales Inv
I_CLOSE_CENIT_LOCAL AT %I0.7: BOOL; /* Control Local Cerrar cenitales In
I_FRIO_LOCAL AT %I0.8: BOOL; /* Control Local Frio Inv. 1 */
I_CALOR_LOCAL AT %I0.9: BOOL; /* Control Local Calor Inv. 1 */
I_FOTOP1_LOCAL AT %I0.10: BOOL; /* Control Local Fotoperiodo 1 Inv.
I_FOTOP2_LOCAL AT %I0.11: BOOL; /* Control Local Fotoperiodo 2 Inv.
I_TERMICOS_OK AT %I0.12: BOOL; /* Termicos OK Inv. 1 */
I_TENSION_24V_OK AT %I0.13: BOOL; /* Tension 24V2 Salidas OK */
/* Salidas */
Q_OPEN_CENIT AT %Q0.0: BOOL; /* Abrir Cenital Inv. 1 */
Q_CLOSE_CENIT AT %Q0.1: BOOL; /* Cerrar Cenital Inv. 1 */
Q_EXTRACTOR AT %Q0.2: BOOL; /* Extractor Inv. 1 */
Q_COOLER AT %Q0.3: BOOL; /* Bomba cooler Inv. 1 */
Q_CALEFACCION AT %Q0.4: BOOL; /* Bomba calefaccion Inv. 1 */
Q_FOTOP1 AT %Q0.5: BOOL; /* Fotoperiodo 1 Inv. 1 */
Q_FOTOP2 AT %Q0.6: BOOL; /* Fotoperiodo 2 Inv. 1 */
Q_FALLO_COM AT %Q0.7: BOOL; /* Fallo Comunicación PLC Inv. 1 */
/* Zona de lectura de PLC */
CENIT_ABRIENDO AT %M0.0: BOOL; /* Cenitales Abriendo Inv.1 */
CENIT_ABIERTOS AT %M0.1: BOOL; /* Cenitales Abiertos Inv.1 */
Line 1, Col 1
Offline program editing

In the lower part of this window we can see the line and column number, the size of the program (in instructions and bytes) if the program can be compiled succesfully, the checksum, and the current viewing mode (offline editing or online monitoring).

After typing in the IEC program, it must be compiled by selecting the option **Compile** in the **Device** menu, or by clicking the button . If the program has errors, these will be displayed in the bottom part of the CRIOSoft window. If not, then we can transfer the program to the board by selecting the **Download** in the **Device** menu, or clicking .

The program can be monitored by entering in **online** mode. But for this to be possible, the program in CRIOSoft and in the SiAMCRIO board must be identical. If they are not, a message will show up to indicate that program online monitoring is not possible, when entering in online mode. In any case, the program window will become read-only and the program will not be modifiable until getting back to offline mode.

If the programs in the PC (CRIOSoft) and PLC (SiAM CRIO) are equal, then the program window will change its appearance when entering online mode, and will look like like this:


Program
Line
Code
Operand
RLO
ACC
244 /*Salta al final si esta en modo aprendizaje
245 LD APREN_COMENZADO 1 0 0
246 JMPC FIN 546 0 0
247
248 /*Si esta abierto poner el tiempo total*/
249 LD I_CEN_OPEN 1 0 0
250 JMPC CEN_OPEN 534 0 0
251
252 /*Si esta cerrado poner el tiempo a 0*/
253 LD I_CEN_CLOSED 1 0 0
254 JMPC CEN_CLOSE 526 0 0
255
256 /*Incrementa Timer si esta abriendo*/
257 LD FLAG_1S 1 0 0
258 AND CENIT_ABRIENDO 1 0 0
259 AND(1 1 0
260 LD T_ACUM_CENIT 0 1 0
261 LT T_TOTAL_CENIT 254 1 0
262) 254 0 0
263 JMPC INCREM_TIMER 493 0 0
264
265 /*Decrementa Timer si esta cerrando*/
266 LD FLAG_1S 1 0 0
267 AND CENIT_CERRANDO 1 0 0
268 AND(1 1 0
269 LD T_ACUM_CENIT 0 1 0
270 GT 0 0 0 0
271) 0 0 0
272 501 0 0
Line 244, Col 1
Size: 171 instructions, 548 bytes
Checksum: 0x4B1D
Online program monitoring

For every program line with executable instructions, we can see:

- Value of the operand.
- Value of the logical accumulator (RLO).
- Value of the numeric accumulator (ACC).

These displayed values are read just in the moment after the execution of the current instruction.

6.10.- Other device options

In the **Program** menu and in the toolbar there are other important options for the programming of the SiAMCRIO board, which are the following:

- **Compile**: Compiles the IEC user program. Any warnings or errors will be displayed in the bottom area of the CRIOSoft window, and double-clicking on them will highlight the code line were the problem was found. It's necessary to compile the program before downloading it to the board.

- **Download**: Select this option to transfer and execute the compiled program to the SiAMCRIO board. Download is only possible in offline mode, and the program must be correctly compiled. If the size of the program is small and fits in the available memory together with the program in execution, then the board will automatically switch to the new program just after download, without the need to stop execution. On the other hand, if the compiled program size is big, then it will be necessary to switch to STOP before download. When the transfer is finished, the user will be asked to switch to RUN mode again to execute the new program.

- **Compare**: This option indicates if the program in the PLC (board) and PC (CRIOSoft IDE) are the same.

- **Memory erase**: This options completely deletes the user program. It can be done in STOP mode.

- **RUN**: Starts the execution of the user program. In the bottom area of the CRIOSoft window will show up a message indicating if the program is successfully started, or if any error has occurred.

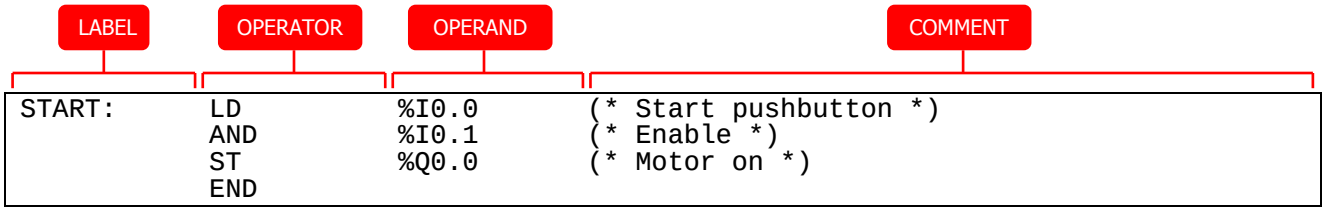
- **STOP**: Stops the execution of the user program. This can be necessary to download a new program, or to erase program memory. Stopping the program will also deactivate all the digital outputs.

7.- IEC PROGRAMMING

7.1.- The instruction list (IL) languange

The *instruction list* (IL) language is the most basic and simple of those defined by the standard IEC 1131-3. It's a low level textual language, whose syntax is similar to assembler. An IL program consists of a sequential list of instructions with an operator and with or without an operand. Every instruction can optionally be preceded by a label, followed by ':'. Comments can appear anywhere in the program, delimited by '(*' y '*)'.

This is an example of an IL program:




```
START:      LD          %I0.0       (* Start pushbutton *)
            AND         %I0.1       (* Enable *)
            ST          %Q0.0       (* Motor on *)
            END
```

The result of the logic operations is stored in two special accumulators:

- RLO (*result of logic operation*) for logic operations (1-bit size).
- ACC (*accumulator*) for arithmetic operations (up to 32-bit size).

The IL operators are overloaded, that is, the same operator can operate on several types of operands, and do different things. So, for example "LD %I0.0" would load the RLO with the value of the input %I0.0. On the other hand, "LD %IW0" would load the ACC with the value of the first inputs word, leaving RLO unchanged.

The ACC type is not fixed, but variable over time. Any load operation will set the ACC type to the variable being loaded, and any subsequent operation on the ACC can change its type. See this example:

```
      LD    1           (* ACC type is INT *)
      LD    1.0         (* ACC type is REAL *)

      LD    1           (* ACC type is INT *)
      ADD   1.0         (* Operand is REAL, so ACC is converted to REAL *)
```

7.2.- Data types

The supported IEC data types are the following:

IEC type	Description	Size in bits	Allowed values
BOOL	Bit	1	TRUE/FALSE
BYTE	Unsigned byte	8	0…255
SINT	Signed byte	8	-128…127
WORD	Unsigned word	16	0…65535
INT	Signed word	16	-32768…32767
DWORD	Unsigned double word	32	0…4294967295
DINT	Signed double word	32	2147483648…+2147483647
REAL	Floating point value (IEEE-754)	32	

7.3.- Instructions

The next table summarizes all the supported instructions. For each instruction, it's shown which data types and which memory areas supports, and also the size in bytes of the instruction. The size of the instruction depends on the data type of the operand, so minimum and maximum sizes are shown. The maximum size of any instruction is 6 bytes.

Besides, for each instruction it's shown if the instruction is a standard IEC instruction or not. Instructions like INC, DEC, CLR… are not standard, but have been included because they allow common operations very easily with only one instruction instead of several.

Some instructions can be used with or without an operand. If used with operand, they perform operations on the operand. If used without operand, they perform the same operations on the RLO or the ACC.

Instruction	IEC	Size (bytes)	Data types								Memory areas						Description
			Bool	Byte	Sint	Word	Int	Dword	Dint	Real	C	I	Q	M	V	S	
Transfer instructions																	
LD	•	3-6	•	•	•	•	•	•	•	•	•	•	•	•	•	•	Load operator value in RLO (if boolean) or in ACC (otherwise).
LDN	•	3-6	•	•	•	•	•	•	•	•	•	•	•	•	•	•	Load negated operator value in RLO (if boolean) or in ACC (otherwise).
ST	•	3-6	•	•	•	•	•	•	•	•		•	•	•	•	•	Store value from RLO (if boolean) or ACC (otherwise) in operand.
STN	•	3-6	•	•	•	•	•	•	•	•		•	•	•	•	•	Store negated value from RLO (if boolean) or ACC (otherwise) in operand.
S[1]	•	1-4	•									•	•	•	•	•	Sets operand to TRUE only if RLO is TRUE. If used without operand, sets RLO to TRUE unconditionally (equivalent to LD TRUE).
R[1]	•	1-4	•									•	•	•	•	•	Sets operand to FALSE only if RLO is TRUE. If used without operand, sets RLO to FALSE unconditionally (equivalent to LD FALSE).
Logic instructions																	
NOT[1]	•	1-4	•	•		•		•				•	•	•	•	•	Performs bitwise inversion of the operand. If used without operand, inverts RLO.
AND	•	3-6	•	•		•		•			•	•	•	•	•	•	Bitwise logical AND operation between operand and RLO (if boolean) or ACC (otherwise).
AND(	•	1															Logic AND between RLO and result of parenthesis[2].
ANDN	•	3-6	•	•		•		•			•	•	•	•	•	•	Bitwise logical AND operation between negated operand and RLO (if boolean) or ACC (otherwise).
ANDN(	•	1															Logical AND between RLO and negated result of parenthesis[2].
OR	•	3-6	•	•		•		•			•	•	•	•	•	•	Bitwise logical OR operation between operand and RLO (if boolean) or ACC (otherwise).
OR(	•	1															Logical OR between RLO and result of parenthesis[2].
ORN	•	3-6	•	•		•		•			•	•	•	•	•	•	Bitwise logical OR operation between negated operand and RLO (if boolean) or ACC (otherwise).

[1] Can be used with or without operand.
[2] Nesting more than 4 parentheses, or reaching the end of the program with any non-closed parenthesis, will cause a fatal error which will stop program execution.

Instruction	IEC	Size (bytes)	Data types								Memory areas						Description
			Bool	Byte	Sint	Word	Int	Dword	Dint	Real	C	I	Q	M	V	S	
ORN(	•	1															Logical OR between RLO and negated result of parenthesis[3].
XOR	•	3-6	•	•		•		•			•	•	•	•	•	•	Bitwise logical XOR operation between operand and RLO (if boolean) or ACC (otherwise).
XOR(	•	1															Logical XOR between RLO and result of parenthesis[3].
XORN	•	3-6	•	•		•		•			•	•	•	•	•	•	Bitwise logical XOR operation between negated operand and RLO (if boolean) or ACC (otherwise).
XORN(	•	1															Logical XOR between RLO and negated result of parenthesis[3].
)	•	1															End of parenthesis operation[4].
Arithmetic instructions																	
ADD	•	3-6		•	•	•	•	•	•	•	•	•	•	•	•	•	Adds ACC to operand and stores result in ACC (ACC = ACC + OP)
SUB	•	3-6		•	•	•	•	•	•	•	•	•	•	•	•	•	Substracts operand from ACC and stores result in ACC (ACC = ACC - OP)
MUL	•	3-6		•	•	•	•	•	•	•	•	•	•	•	•	•	Multiplies operand with ACC and stores result in ACC (ACC = ACC * OP)
DIV	•	3-6		•	•	•	•	•	•	•	•	•	•	•	•	•	Divides ACC by operand and stores result in ACC (ACC = ACC / OP)[5].
MOD	•	3-6		•	•	•	•	•	•		•	•	•	•	•	•	Divides ACC by operand and stores remainder in ACC (ACC = ACC % OP)[5].
INC[6]		1-4		•	•	•	•	•	•			•	•	•	•	•	Increments operand (OP = OP + 1). If used without operand, increments ACC (ACC = ACC + 1).
DEC[6]		1-4		•	•	•	•	•	•			•	•	•	•	•	Decrements operand (OP = OP - 1). If used without operand, decrements ACC (ACC = ACC - 1).
CLR[6]		1-4		•	•	•	•	•	•			•	•	•	•	•	Sets operand to 0 (OP = 0). If used without operand, sets ACC to 0 (ACC = 0).

[3] Nesting more than 4 parentheses, or reaching the end of the program with any non-closed parenthesis, will cause a fatal error which will stop program execution.
[4] Parenthesis must be open before being closed, otherwise a fatal error will stop program execution.
[5] If operand is 0, a fatal error will stop program execution.
[6] Can be used with or without operand.

Instruction	IEC	Size (bytes)	Data types								Memory areas						Description
			Bool	Byte	Sint	Word	Int	Dword	Dint	Real	C	I	Q	M	V	S	
SHL[7]	•	1-4		•	•	•	•	•	•			•	•	•	•	•	Shifts operand one bit to the left (OP = OP << 1). If used without operand, shifts ACC (ACC = ACC << 1).
SHR[7]	•	1-4		•	•	•	•	•	•			•	•	•	•	•	Shifts operand one bit to the right (OP = OP >> 1). If used without operand, shifts ACC (ACC = ACC >> 1).
Compare instructions																	
GT	•	3-6		•	•	•	•	•	•	•	•	•	•	•	•	•	Sets RLO to TRUE if ACC is greater than the operand. Otherwise sets RLO to FALSE.
GE	•	3-6		•	•	•	•	•	•	•	•	•	•	•	•	•	Sets RLO to TRUE if ACC is greater or equal than the operand. Otherwise sets RLO to FALSE.
EQ	•	3-6		•	•	•	•	•	•	•	•	•	•	•	•	•	Sets RLO to TRUE if ACC is equal to the operand. Otherwise sets RLO to FALSE.
NE	•	3-6		•	•	•	•	•	•	•	•	•	•	•	•	•	Sets RLO to TRUE if ACC is not equal to the operand. Otherwise sets RLO to FALSE.
LE	•	3-6		•	•	•	•	•	•	•	•	•	•	•	•	•	Sets RLO to TRUE if ACC is lower or equal than the operand. Otherwise sets RLO to FALSE.
LT	•	3-6		•	•	•	•	•	•	•	•	•	•	•	•	•	Sets RLO to TRUE if ACC is lower than the operand. Otherwise sets RLO to FALSE.
Program control instructions																	
JMP	•	4				•					•						Jumps unconditionally to address indicated by the operand.
JMPC	•	4				•					•						Jumps conditionally (when RLO is TRUE) to address indicated by the operand.
JMPCN	•	4				•					•						Jumps conditionally (when RLO is FALSE) to address indicated by the operand.
CAL[8]	•	4				•					•						Calls unconditionally subroutine in address indicated by the operand.
CALC[8]	•	4				•					•						Calls conditionally (when RLO is TRUE) subroutine in address indicated by the operand.
CALCN[8]	•	4				•					•						Calls conditionally (when RLO is FALSE) subroutine in address indicated by the operand.

[7] Can be used with or without operand.
[8] If more than 4 calls are nested, a fatal error will stop program execution.

Instruction	IEC	Size (bytes)	Data types								Memory areas						Description
			Bool	Byte	Sint	Word	Int	Dword	Dint	Real	C	I	Q	M	V	S	
LOOP		4				•					•						Decrements ACC and jumps to address if ACC is 0.
LOOPN		4				•					•						Decrements ACC and jumps to address if ACC is not 0.
RET[9]	•	1															Ends subroutine unconditionally and returns to instruction after CAL.
RETC[9]	•	1															Ends subroutine conditionally (when RLO is TRUE) and returns to instruction after CAL.
RETCN[9]	•	1															Ends subroutine conditionally (when RLO is FALSE) and returns to instruction after CAL.
END	•	1															End of user program. Its use is compulsory.
Miscellaneous instructions																	
NOP		1															No operation.
RTRIG		1-4	•											•	•		Detects rising edge in RLO, and stores previous result in operand.
FTRIG		1-4	•											•	•		Detects falling edge in RLO, and stores previous result in operand.
Accumulator instructions																	
SWAP		1															Swap bytes in ACC.
ABS		1															Absolute value of ACC.
SQRT		1															Square root of ACC.
LN		1															Natural logarithm of ACC.
LOG		1															Base-10 logarithm of ACC.
EXP		1															Exponential of ACC.
SIN		1															Sine of ACC.
COS		1															Cosine of ACC.
TAN		1															Tangent of ACC.
ASIN		1															Arc sine of ACC.
ACOS		1															Arc cosine of ACC.
ATAN		1															Arc tangent of ACC.

[9] If it's called outside a subroutine, fatal error will stop program execution.

Instruction	IEC	Size (bytes)	Data types								Memory areas						Description
			Bool	Byte	Sint	Word	Int	Dword	Dint	Real	C	I	Q	M	V	S	
Conversion instructions																	
TO_BYTE		1															Converts ACC to BYTE.
TO_SINT		1															Converts ACC to SINT.
TO_WORD		1															Converts ACC to WORD.
TO_INT		1															Converts ACC to INT.
TO_DWORD		1															Converts ACC to DWORD.
TO_DINT		1															Converts ACC to DINT.
TO_REAL		1															Converts ACC to REAL.
BCD		1															Converts ACC from decimal to BCD.
I_BCD		1															Converts ACC from BCD to decimal.

7.4.- Memory areas

The operands which follow the instructions can be either constant values, or belong to one of these memory areas:

Area	Description	Size (words)	Range
I	Inputs memory	9	%IW0…%IW8
Q	Outputs memory	1	%QW0
M	RAM user memory (non-persistent)	256	%MW0…%MW255
V	EEPROM user memory (persistent)	256	%VW0…%VW255
S	System memory	256	%SW0…%SW255

Reading registers outside the values displayed in this table (for example %IW9) will cause no error, but will always return 0. Accordingly, writing to any non-valid register will cause no change at all.

As can be shown in the instructions table, some instructions like ST, S, R… can operate on inputs memory. This can be useful to simulate or force signals in some situations.

The system memory is mostly a read-only area. Writing on most registers will cause no effect (and no error as well). The other areas have read/write access.

The V memory (persistent) is really in the RAM memory of the microcontroller, but its periodically written to the EEPROM. The number of writings in the EEPROM is limited and must be minimized in order to maximize the endurance of this non-volatile memory. Because of this, the dumping of V memory to EEPROM is performed once per hour. This big time can cause the loss of important data if a power failure happens before the dumping takes place. To avoid this, there is a system bit (%S1.8) that can be set by the user program to force an instantaneous dumping of the V memory to EEPROM and ensure critical data.

This dumping is done in a smart way to maximize EEPROM endurance, by writing only registers that have been modified since the previous dump.

The integrity of the V memory is checked at start-up, and it's mapped to system bit %S0.7. If this bit is active, EEPROM memory can be corrupted and the data stored in V memory may not be valid anymore. The user program should check this bit on start-up and restore default values for the application if it's active, or signal an alarm and wait for user intervention –for example, by externally downloading of parameters via modbus- before starting the main program.

7.4.1.- Inputs memory

<table>
<tr><th>Register</th><th>Description</th></tr>
<tr><td>%IW0</td><td>Digital inputs register (read/write):

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>Digital input DI0</td></tr>
<tr><td>1</td><td>Digital input DI1</td></tr>
<tr><td>2</td><td>Digital input DI2</td></tr>
<tr><td colspan="2">…</td></tr>
<tr><td>15</td><td>Digital input DI15</td></tr>
</table>

If any input is configured as analog, the corresponding bit will be active if the analog value is above 50% of the maximum analog level (5V / 20mA), and inactive otherwise.</td></tr>
<tr><td>%IW1…%IW8</td><td>Scaled analog input AI0…AI7 (read/write)
SINT type value with range -32768…32767.</td></tr>
</table>

7.4.2.- Outputs memory

<table>
<tr><th>Register</th><th>Description</th></tr>
<tr><td>%QW0</td><td>Digital outputs (read/write):

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>Digital output DI0</td></tr>
<tr><td>1</td><td>Digital output DI1</td></tr>
<tr><td>2</td><td>Digital output DI2</td></tr>
<tr><td colspan="2">…</td></tr>
<tr><td>11</td><td>Digital output DI11</td></tr>
<tr><td>12</td><td>Not used</td></tr>
<tr><td>13</td><td>Not used</td></tr>
<tr><td>14</td><td>Not used</td></tr>
<tr><td>15</td><td>Not used</td></tr>
</table>
</td></tr>
</table>

7.4.3.- RAM memory

Register	Description
%MW0…%MW255	RAM user memory (read/write). There registers are non-persistent, and will be lost when power is removed from the board.

7.4.4.- EEPROM memory

Register	Description
%VW0…%VW255	EEPROM user memory (read/write). There registers are persistent, and will not be lost even when power is removed from the board.

7.4.5.- System memory

<table>
<tr><th>Register</th><th>Description</th></tr>
<tr><td>%SW0</td><td>General status register (read-only):
<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>%S0.0</td><td>Bit always FALSE.</td></tr>
<tr><td>%S0.1</td><td>Bit always TRUE.</td></tr>
<tr><td>%S0.2</td><td>First cycle. TRUE in the first PLC scan after switching from STOP to RUN, and FALSE in all the others.</td></tr>
<tr><td>%S0.3</td><td>Not used.</td></tr>
<tr><td>%S0.4</td><td>This bit indicates if last reset was a power-on (normal) reset.</td></tr>
<tr><td>%S0.5</td><td>This bit indicates if last reset was a brown-out reset.</td></tr>
<tr><td>%S0.6</td><td>This bit indicates if last reset was a watchdog reset.</td></tr>
<tr><td>%S0.7</td><td>EEPROM checksum error. The persistent memory is checked on start-up, and this bit is set if it’s not correct. The user program should check it to load default values for the important variables in the program.</td></tr>
<tr><td>%S0.8</td><td>Program memory checksum error. The program memory is checked when switching to RUN mode. If it’s not correct, this bit is set and the PLC remains in STOP mode.</td></tr>
<tr><td>%S0.9</td><td>EEPROM memory changed. This bit indicates if there is any change in V memory pending to be written to EEPROM. The V memory is periodically dumped to EEPROM, but the user program can check this bit to force an instantaneous dump to ensure the persistence of critical data.</td></tr>
<tr><td>%S0.10</td><td>RUN mode. This bit is active when the user program is executing (RUN mode), and inactive in STOP mode.</td></tr>
<tr><td>%S0.11</td><td>Reserved.</td></tr>
<tr><td>%S0.12</td><td>Reserved.</td></tr>
<tr><td>%S0.13</td><td>Not used.</td></tr>
<tr><td>%S0.14</td><td>Not used.</td></tr>
<tr><td>%S0.15</td><td>Not used.</td></tr>
</table></td></tr>
<tr><td>%SW1</td><td>General command register (read/write):
<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>%S1.0</td><td>Reset faults.</td></tr>
<tr><td>%S1.1</td><td>Reset microcontroller (software reset).</td></tr>
<tr><td>%S1.2</td><td>Reset maximum cycle time.</td></tr>
<tr><td>%S1.3</td><td>Start program execution (RUN mode request).</td></tr>
<tr><td>%S1.4</td><td>Not used.</td></tr>
<tr><td>%S1.5</td><td>Reserved.</td></tr>
<tr><td>%S1.6</td><td>Reserved.</td></tr>
<tr><td>%S1.7</td><td>Enable forcing of inputs and outputs (see %SW20…%SW23)</td></tr>
<tr><td>%S1.8</td><td>Force V memory dump to EEPROM. The dump will not take place immediately after settingo this bit, but at the end of the PLC scan.</td></tr>
<tr><td>%S1.8</td><td>Not used.</td></tr>
<tr><td>%S1.9</td><td>Not used.</td></tr>
<tr><td>%S1.10</td><td>Not used.</td></tr>
<tr><td>%S1.11</td><td>Not used.</td></tr>
<tr><td>%S1.12</td><td>Not used.</td></tr>
<tr><td>%S1.13</td><td>Not used.</td></tr>
<tr><td>%S1.14</td><td>Not used.</td></tr>
<tr><td>%S1.15</td><td>Not used.</td></tr>
</table></td></tr>
</table>

<table>
<tr><th>Register</th><th>Description</th></tr>
<tr><td>%SW2</td><td>Time flags register (read-only):

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>%S2.0</td><td>Flag active one scan every 100 ms.</td></tr>
<tr><td>%S2.1</td><td>Flag active one scan every 200 ms.</td></tr>
<tr><td>%S2.2</td><td>Flag active one scan every 500 ms.</td></tr>
<tr><td>%S2.3</td><td>Flag active one scan every 1 second.</td></tr>
<tr><td>%S2.4</td><td>Flag active one scan every 2 seconds.</td></tr>
<tr><td>%S2.5</td><td>Flag active one scan every 5 seconds.</td></tr>
<tr><td>%S2.6</td><td>Flag active one scan every 10 seconds.</td></tr>
<tr><td>%S2.7</td><td>Flag active one scan every 15 seconds.</td></tr>
<tr><td>%S2.8</td><td>Flag active one scan every 30 seconds.</td></tr>
<tr><td>%S2.9</td><td>Flag active one scan every 1 minute.</td></tr>
<tr><td>%S2.10</td><td>Flag active one scan every 1 hour.</td></tr>
<tr><td>%S2.11</td><td>Flag active one scan every 1 day.</td></tr>
<tr><td>%S2.12</td><td>Not used.</td></tr>
<tr><td>%S2.13</td><td>Not used.</td></tr>
<tr><td>%S2.14</td><td>Not used.</td></tr>
<tr><td>%S2.15</td><td>Not used.</td></tr>
</table>
</td></tr>
<tr><td>%SW3</td><td>Time pulses register (read-only):

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>%S3.0</td><td>Pulse with 100 ms period (10 Hz).</td></tr>
<tr><td>%S3.1</td><td>Pulse with 200 ms period (5 Hz).</td></tr>
<tr><td>%S3.2</td><td>Pulse with 500 ms period (2 Hz).</td></tr>
<tr><td>%S3.3</td><td>Pulse with 1 second period (1 Hz).</td></tr>
<tr><td>%S3.4</td><td>Pulse with 2 seconds period (0.5 Hz).</td></tr>
<tr><td>%S3.5</td><td>Pulse with 5 seconds period (0.2 Hz).</td></tr>
<tr><td>%S3.6</td><td>Pulse with 10 seconds period (0.1 Hz).</td></tr>
<tr><td>%S3.7</td><td>Pulse with 15 seconds period.</td></tr>
<tr><td>%S3.8</td><td>Pulse with 30 seconds period.</td></tr>
<tr><td>%S3.9</td><td>Pulse with 1 minute period.</td></tr>
<tr><td>%S3.10</td><td>Pulse with 1 hour period.</td></tr>
<tr><td>%S3.11</td><td>Pulse with 1 day period.</td></tr>
<tr><td>%S3.12</td><td>Not used.</td></tr>
<tr><td>%S3.13</td><td>Not used.</td></tr>
<tr><td>%S3.14</td><td>Not used.</td></tr>
<tr><td>%S3.15</td><td>Not used.</td></tr>
</table>
</td></tr>
<tr><td>%SW4</td><td>Milliseconds counter (read-only). Starts at 0 when the board is powered-up, and increments every millisecond, overflowing at 65535.

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>%S4.0</td><td>Pulse with 1 ms period (1000 Hz).</td></tr>
<tr><td>%S4.1</td><td>Pulse with 2 ms period (500 Hz).</td></tr>
<tr><td>%S4.2</td><td>Pulse with 4 ms period (250 Hz).</td></tr>
<tr><td>%S4.3</td><td>Pulse with 8 ms period (125 Hz).</td></tr>
<tr><td>…</td><td></td></tr>
<tr><td>%S4.14</td><td>Pulse with 16384 ms period</td></tr>
<tr><td>%S4.15</td><td>Pulse with 32768 ms period</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Register</th><th>Description</th></tr>
<tr><td>%SW5</td><td>Scan cycles counter (read-only). Starts at 0 when board enters RUN mode, and increments every scan cycle, overflowing at 65535.</td></tr>
<tr><td>%SW6</td><td>Configuration switches (read-only). On startup, the configuration dip-switches are mapped to this register (see section 5).

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>%S6.0</td><td>Modbus address bit 0 (see section 5.1)</td></tr>
<tr><td>%S6.1</td><td>Modbus address bit 1.</td></tr>
<tr><td>%S6.2</td><td>Modbus address bit 2.</td></tr>
<tr><td>%S6.3</td><td>Modbus address bit 3</td></tr>
<tr><td>%S6.4</td><td>Modbus address bit 4.</td></tr>
<tr><td>%S6.5</td><td>Modbus address bit 5.</td></tr>
<tr><td>%S6.6</td><td>Modbus address bit 6.</td></tr>
<tr><td>%S6.7</td><td>Modbus address bit 7.</td></tr>
<tr><td>%S6.8</td><td>Baud rate switch S1 (see section 5.2).</td></tr>
<tr><td>%S6.9</td><td>Baud rate switch S2.</td></tr>
<tr><td>%S6.10</td><td>Baud rate switch S3.</td></tr>
<tr><td>%S6.11</td><td>User switch (see section 5.3).</td></tr>
<tr><td>%S6.12</td><td>Not used.</td></tr>
<tr><td>%S6.13</td><td>Not used.</td></tr>
<tr><td>%S6.14</td><td>Not used.</td></tr>
<tr><td>%S6.15</td><td>Not used.</td></tr>
</table></td></tr>
<tr><td>%SW7</td><td>Last scan cycle time in milliseconds (read-only).</td></tr>
<tr><td>%SW8</td><td>Maximum scan cycle time in milliseconds (read-only). Reset by setting %S1.2.</td></tr>
<tr><td>%SW9…%SW10</td><td>Reserved.</td></tr>
<tr><td>%SW11</td><td>Program counter (read-only).</td></tr>
<tr><td>%SW12</td><td>Program error code (read-only, see section 7.5).</td></tr>
<tr><td>%SW13</td><td>Program counter for instruction with error (read-only).</td></tr>
<tr><td>%SW14</td><td>Counter of number of nested calls (read-only).</td></tr>
<tr><td>%SW15</td><td>Counter of number of nested parentheses (read-only).</td></tr>
<tr><td>%SW16…%SW19</td><td>Reserved.</td></tr>
<tr><td>%SW20</td><td>Inputs force OFF mask (read/write). When I/O forces are active (%S1.7), every active bit in this register will cause the corresponding input to be forced OFF, irrespectively of the voltage in the physical input connector.</td></tr>
<tr><td>%SW21</td><td>Inputs force ON mask (read/write). When I/O forces are active (%S1.7), every active bit in this register will cause the corresponding input to be forced ON, irrespectively of the voltage in the physical input connector.</td></tr>
<tr><td>%SW22</td><td>Outputs force OFF mask (read/write). When I/O forces are active (%S1.7), every active bit in this register will cause the corresponding output to be forced OFF, irrespectively of the result of the user program for that output.</td></tr>
<tr><td>%SW23</td><td>Outputs force ON mask (read/write). When I/O forces are active (%S1.7), every active bit in this register will cause the corresponding output to be forced ON, irrespectively of the result of the user program for that output.</td></tr>
<tr><td>%SW30…%SW37</td><td>Unscaled analog input AI0…AI7 (read-only).
WORD type value with range 0…1023.</td></tr>
</table>

<table>
<tr><th>Register</th><th>Description</th></tr>
<tr><td>%SW40…%SW44</td><td>Indirect addressing pointers to M memory area (read/write).
These pointers allow accessing M memory through virtual registers %SW50…%SW54. For example, if we store the value 10 in %SW40, then when we access virtual register %SW50 we are really working with real register %MW10. This is very useful for some operations like loops.</td></tr>
<tr><td>%SW45…%SW49</td><td>Indirect addressing pointers to V memory area (read/write).
These pointers allow accessing V memory through virtual registers %SW55…%SW59. For example, if we store the value 10 in %SW45, then when we access virtual register %SW55 we are really working with real register %VW10. This is very useful for some operations like loops.</td></tr>
<tr><td>%SW50…%SW54</td><td>Virtual registers for indirect addressing in M memory area (read/write).</td></tr>
<tr><td>%SW55…%SW59</td><td>Virtual registers for indirect addressing in V memory area (read/write).</td></tr>
<tr><td>%SW60</td><td>Serial port status register (read-only):

<table><tr><th>Bit</th><th>Description</th></tr><tr><td>%S60.0</td><td>RX Timeout.</td></tr><tr><td>%S60.1</td><td>RX Buffer overrun.</td></tr><tr><td>%S60.2</td><td>RX Framing error.</td></tr><tr><td>%S60.3</td><td>RX Parity error.</td></tr><tr><td>%S60.4</td><td>RX Buffer full.</td></tr><tr><td>%S60.5</td><td>Not used.</td></tr><tr><td colspan="2">…</td></tr><tr><td>%S60.15</td><td>Not used.</td></tr></table></td></tr>
<tr><td>%SW61</td><td>Serial port command register (read/write):

<table><tr><th>Bit</th><th>Description</th></tr><tr><td>%S61.0</td><td>Reset serial port errors in %SW60.</td></tr><tr><td>%S61.1</td><td>Not used.</td></tr><tr><td colspan="2">…</td></tr><tr><td>%S61.15</td><td>Not used.</td></tr></table></td></tr>
<tr><td>%SW62</td><td>Serial port baud rate (read/write). On startup, this setting is read from dip-switches (see section 5.2), but can be overwritten by user program at any time. The possible values are:

<table><tr><th>Value</th><th>Baud rate (bps)</th></tr><tr><td>0</td><td>4800</td></tr><tr><td>1</td><td>9600</td></tr><tr><td>2</td><td>19200</td></tr><tr><td>3</td><td>28800</td></tr><tr><td>4</td><td>38400</td></tr><tr><td>5</td><td>57600</td></tr><tr><td>6</td><td>115200</td></tr><tr><td>7</td><td>128000</td></tr></table></td></tr>
<tr><td>%SW63</td><td>Serial port timeout in milliseconds (read/write).</td></tr>
<tr><td>%SW64</td><td>Serial port received bytes (read-only). Indicates the number of bytes received in the serial port buffer.</td></tr>
</table>

<table>
<tr><th>Register</th><th>Description</th></tr>
<tr><td>%SW70</td><td>Modbus status register (read-only):

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>%S70.0</td><td>Link error. This bit becomes active when there is no communication between the board and the modbus master in some time (modbus link timeout, see section 6.8.1).</td></tr>
<tr><td>%S70.1</td><td>Frame control error.</td></tr>
<tr><td>%S70.2</td><td>Frame length error.</td></tr>
<tr><td>%S70.3</td><td>Checksum error.</td></tr>
<tr><td>%S70.4</td><td>Timeout error.</td></tr>
<tr><td>%S70.5</td><td>Not used.</td></tr>
<tr><td colspan="2">…</td></tr>
<tr><td>%S70.15</td><td>Not used.</td></tr>
</table></td></tr>
<tr><td>%SW71</td><td>Modbus command register (read/write):

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>%S71.0</td><td>Reset modbus errors in %SW70.</td></tr>
<tr><td>%S71.1</td><td>Reset modbus diagnostic counters.</td></tr>
<tr><td>%S71.2</td><td>Not used.</td></tr>
<tr><td colspan="2">…</td></tr>
<tr><td>%S71.15</td><td>Not used.</td></tr>
</table></td></tr>
<tr><td>%SW72</td><td>Modbus slave address (read/write). On startup, this setting is read from dip-switches (see section 5.1), but can be overwritten by user program at any time. Valid values are 1…246.</td></tr>
<tr><td>%SW73</td><td>Modbus initial delay (read/write). On startup, the board waits for this time (in milliseconds) before starting communications.</td></tr>
<tr><td>%SW74</td><td>Modbus reply delay (read/write). After receiving a modbus request, the board waits for this time (in milliseconds) before sending the reply.</td></tr>
<tr><td>%SW75</td><td>Board model ID (read only). This value identifies the model of the board (7001 for this model).</td></tr>
<tr><td>%SW76</td><td>Hardware version (read only).</td></tr>
<tr><td>%SW77</td><td>Firmware version (read only).</td></tr>
<tr><td>%SW80…%SW81</td><td>Modbus diagnostic counter for frames detected in bus (read only).
This is a 32 bit counter (%SW80 high word; %SW81 low word).
Can be reset by setting bit %S71.1</td></tr>
<tr><td>%SW82…%SW83</td><td>Modbus diagnostic counter for frames addressed to slave (read only).
This is a 32 bit counter (%SW82 high word; %SW83 low word).
Can be reset by setting bit %S71.1</td></tr>
<tr><td>%SW84…%SW85</td><td>Modbus diagnostic counter for incomplete frames addressed to slave (read only).
This is a 32 bit counter (%SW84 high word; %SW85 low word).
Can be reset by setting bit %S71.1</td></tr>
<tr><td>%SW86…%SW87</td><td>Modbus diagnostic counter for frames addressed to slave with checksum errors (read only). This is a 32 bit counter (%SW86 high word; %SW87 low word).
Can be reset by setting bit %S71.1</td></tr>
<tr><td>%SW88…%SW89</td><td>Modbus diagnostic counter for frames addressed to slave with parameter errors (read only). This is a 32 bit counter (%SW88 high word; %SW89 low word).
Can be reset by setting bit %S71.1</td></tr>
<tr><td>%SW90…%SW91</td><td>Modbus diagnostic counter for frames addressed to slave not supported (read only). This is a 32 bit counter (%SW90 high word; %SW91 low word).
Can be reset by setting bit %S71.1</td></tr>
</table>

<table>
<tr><th>Register</th><th>Description</th></tr>
<tr><td>%SW92…%SW93</td><td>Modbus diagnostic counter for frames sent by slave (read only).
This is a 32 bit counter (%SW90 high word; %SW91 low word).
Can be reset by setting bit %S71.1</td></tr>
<tr><td>%SW100</td><td>Diagnostic LED error code (read only):

<table><tr><th>Value</th><th>Description</th></tr><tr><td>0</td><td>CPU in STOP mode (no errors)</td></tr><tr><td>1</td><td>CPU in RUN mode (no errors nor warnings)</td></tr><tr><td>2</td><td>CPU in RUN mode with warnings</td></tr><tr><td>3</td><td>CPU in STOP mode with error</td></tr></table></td></tr>
<tr><td>%SW110</td><td>Time flags status register (read-only). This register is equal to %SW2, but runs asynchronously to the main scan cycle.</td></tr>
<tr><td>%SW111</td><td>Time command register (read/write):

<table><tr><th>Bit</th><th>Description</th></tr><tr><td>%S111.0</td><td>Reset running time counters.</td></tr><tr><td>%S111.1</td><td>Reserved.</td></tr><tr><td>%S111.2</td><td>Not used.</td></tr><tr><td colspan="2">…</td></tr><tr><td>%S111.15</td><td>Not used.</td></tr></table></td></tr>
<tr><td>%SW112</td><td>Running time counter that increments every 100 milliseconds, from 0 to 9 (read-only). Can be reset by setting %S111.0.</td></tr>
<tr><td>%SW113</td><td>Running time counter that increments every second, from 0 to 59 (read-only). Can be reset by setting %S111.0.</td></tr>
<tr><td>%SW114</td><td>Running time counter that increments every minute, from 0 to 59 (read-only). Can be reset by setting %S111.0.</td></tr>
<tr><td>%SW115</td><td>Running time counter that increments every hour, from 0 to 23 (read-only). Can be reset by setting %S111.0.</td></tr>
<tr><td>%SW116</td><td>Running time counter that increments every day (read-only). Can be reset by setting %S111.0.</td></tr>
<tr><td>%SW120…%SW127</td><td>Reserved.</td></tr>
<tr><td>%SW130…%SW137</td><td>Lower limit of scaling for analog inputs AI0…AI7 (read/write). See section 6.8.3.</td></tr>
<tr><td>%SW140…%SW147</td><td>Highet limit of scaling for analog inputs AI0…AI7 (read/write). See section 6.8.3.</td></tr>
<tr><td>%SW150</td><td>Modbus link timeout in milliseconds (read/write). See section 6.8.1.</td></tr>
</table>

7.5.- Program errors and warnings

There are some situations where the user program can not be correctly executed. This case will be visually indicated by a different blinking of the board's status LED (see section 4.1), but for a detailed description of the error it's necessary to connect online and open the device status window (see section 6.7).

All the possible errors and warnings are defined by a numeric code that is mapped to system register %SW12. A value of 0 indicates no error or warning. Any value between 1 and 99 indicates a warning, and any value of 100 or higher indicates an error.

The possible values and error causes are these:

Code	Description
0	Program execution OK (no error; no warning)
1	Program empty. This indicates that the board was requested to enter into RUN mode with no program downloaded.
2	Type conversion with possible information loss.
3	Invalid accumulator type for instruction.
4	Math operation not possible.
5	Division by zero. If any DIV or MOD is executed when the ACC is zero, this warning will appear.
100	Maximum time exceeded (watchdog). If the user program execution time is longer than the specified watchdog timeout (see section 6.8.1) then execution is stopped and this error will appear.
101	Program checksum error. The board could not switch to RUN mode because it detected errors in the downloaded user program. The program should be downloaded again to correct this error.
102	Operation code not valid. All the possible instructions (see section 7.3) have a numeric code. If the board detects an unknown code, it stops execution and displays this error. Normally this is checked by the compiler, so this error normally indicates a problem in program memory.
103	Operator area not valid. Not all operator areas are valid for every instruction. For example, RTRIG or FTRIG instructions can only operate on M and V memory. If some invalid combination is found, this error stops program execution. Normally this is checked by the compiler, so this error normally indicates a problem in program memory.
104	Operator type not valid. Not all operator types are valid for every instruction. For example, S or R instructions can only operate on BOOLs. If some invalid combination is found, this error stops program execution. Normally this is checked by the compiler, so this error normally indicates a problem in program memory.
105	Closure of non-open parenthesis. The ')' instruction must be used only after opening the parenthesis with 'AND(', 'OR(' or 'XOR('. Otherwise, this error will stop program execution.
106	Too many open parentheses. No more than 4 parentheses can be nested.
107	Too many nested calls. No more than 4 CALs can be nested.
108	Jump to wrong address.
109	RET without CAL.
110	End of program with open parenthesis. All open parentheses must be closed before reaching the end of the program.

111	End of program inside subroutine. The END instruction must be used only in the main program level, and n
112	Instruction not allowed inside parenthesis. Program control instructions (JMP, CAL, RET, LOOP) can not be used inside parentheses.

7.6.- Programming examples

7.6.1.- Simple automatic door

The next example program would control a motorized door or blind, with two pushbuttons (*raise* and *lower*), and two limit switches (*up* and *down*).

```
VAR_GLOBAL

      (* Inputs *)
      PB_RAISE AT %I0.0: BOOL;      (* Raise pushbutton *)
      PB_LOWER AT %I0.1: BOOL;      (* Lower pushbutton *)
      SWITCH_UP AT %I0.2: BOOL;     (* Limit switch up *)
      SWITCH_DOWN AT %I0.3: BOOL;   (* Limit switch down *)

      (* Outputs *)
      RAISE AT %Q0.0: BOOL;         (* Raise command *)
      LOWER AT %Q0.1: BOOL;         (* Lower command *)

END_VAR

PROGRAM Main

      LD    PB_RAISE
      ANDN  SWITCH_UP
      ST    RAISE

      LD    PB_LOWER
      ANDN  SWITCH_DOWN
      ANDN  RAISE
      ST    LOWER

      END

END_PROGRAM
```

7.6.2.- Automatic door with local and remote control

Now the automatic blind has a *local* control by pushbuttons, and a *remote* control by a modbus master like a PLC o SCADA software. The selection between local and remote mode is done with a two-position selector connected to another input.

```
VAR_GLOBAL

      (* Inputs *)
      PB_RAISE AT %I0.0: BOOL;      (* Raise pushbutton *)
      PB_LOWER AT %I0.1: BOOL;      (* Lower pushbutton *)
      SWITCH_UP AT %I0.2: BOOL;     (* Limit switch up *)
      SWITCH_DOWN AT %I0.3: BOOL;   (* Limit switch down *)
      LOCAL AT %I0.4: BOOL;         (* Local/Remote control selector *)

      (* Remote commands *)
      REMOTE_RAISE AT %I0.0: BOOL;  (* Remote raise command *)
      REMOTE_LOWER AT %I0.1: BOOL;  (* Remote lower command *)

      (* Outputs *)
      RAISE AT %Q0.0: BOOL;         (* Raise command *)
      LOWER AT %Q0.1: BOOL;         (* Lower command *)

END_VAR

PROGRAM Main

      LD    LOCAL
      AND   PB_RAISE
      OR(
            LDN   LOCAL
            AND   REMOTE_RAISE
      )
      ANDN  SWITCH_UP
      ST    RAISE

      LD    LOCAL
      AND   PB_LOWER
      OR(
            LDN   LOCAL
            AND   REMOTE_LOWER
      )
      ANDN  SWITCH_DOWN
      ANDN  RAISE
      ST    LOWER

      END

END_PROGRAM
```

Instead of the local/remote selector, we could use the link error bit (%S70.0), in such a way that if communications are active (link error OFF) the control would be remote, but if communication is lost (link error ON) the program would automatically switch to local control.

7.6.3.- Timers

There are no instructions for direct manipulation of timers, but these are very easy to program by using the time flag bits found in %SW2. For example, the following code activates output %Q0.0 when input %I0.0 is active for 5 seconds.

```
VAR_GLOBAL

      IN0 AT %I0.0: BOOL;
      OUT0 AT %Q0.0: BOOL;
      PULSE_1S AT %S2.3: BOOL;
      TIMER AT %MW0: WORD;

END_VAR

PROGRAM Main

            LD    IN0          (* Check the input *)
            JMPC  COUNT        (* If it's 1, let's check the time flag *)
            CLR   TIMER        (* If it's 0, reset the timer *)
            JMP   CHECK

COUNT:      LDN   PULSE_1S     (* Check the time flag *)
            JMPC  CHECK        (* If it's 0, don't increment the timer *)
            INC   TIMER        (* If it's 1, increment the timer *)

CHECK:      LD    TIMER        (* If timer... *)
            GE    5            (* ...is greater than 5 seconds... *)
            ST    OUT0         (* ...then we activate the output *)

            END

END_PROGRAM
```

7.6.4.- Intermitencies

There are many system bits which can be used for regular intermittencies, as shown in this example:

```
PROGRAM Main

      LD    %S3.0       (* Intermitence 10 Hz *)
      ST    %Q0.0

      LD    %S3.3       (* Intermitence 1 Hz *)
      ST    %Q0.1

      LD    %S3.9       (* Intermitence 1/60 Hz *)
      ST    %Q0.2

      LD    %S4.8       (* Intermitence every 256ms (4 Hz approx.) *)
      ST    %Q0.2

      LD    %S4.9       (* Intermitence every 512ms (2 Hz approx.) *)
      ST    %Q0.2

      LD    %S4.11      (* Intermitence every 2048ms (0.5 Hz approx.) *)
      ST    %Q0.2

END_PROGRAM
```

If an irregular intermitente is needed, then it's necessary to work with counters, as shown in this example that activates an output for 10 seconds and deactivates it for 5 seconds.

```
VAR_GLOBAL

      OUT0 AT %Q0.0: BOOL;
      PULSE_1S AT %S2.3: BOOL;
      COUNT AT %MW0: WORD;

END_VAR

PROGRAM Main

            LDN   PULSE_1S    (* Check 1 second pulse *)
            JMPC  LABEL1
            INC   COUNT       (* Increment counter *)

LABEL1:     LD    TRUE
            LD    COUNT       (* Check counter *)
            LT    15
            JMPC  LABEL2
            CLR   COUNT       (* Reset if greater than 15 (10+5) seconds *)

LABEL2:     LD    TRUE
            LD    COUNT
            LT    10
            ST    OUT0        (* Activate output if counter < 10 seconds *)

END_PROGRAM
```

7.6.5.- EEPROM checksum management

The following example would restore default values for the variables, in the event of a checksum error on program startup:

```
VAR_GLOBAL

      (* System bits *)
      FIRST_CYCLE AT %S0.2: BOOL;
      EEPROM_ERROR AT %S0.7: BOOL;
      EEPROM_WRITE AT %S1.8: BOOL;
      EEPROM_RESTORED AT %M0.0: BOOL;

      (* Variables in V memory *)
      VAR1: INT;
      VAR2: INT;
      VAR3: INT;
      ...

END_VAR

PROGRAM Main

            LD    FIRST_CYCLE       (* Check first PLC scan *)
            AND   EEPROM_ERROR      (* If EEPROM checksum error... *)
            JMPC  RESET_EE          (* ...then jump to restore values *)
            JMP   MAIN_PROG         (* ...otherwise go to main program *)

RESET_EE:   LD    100               (* Default value for VAR1 *)
            ST    VAR1
            LD    200               (* Default value for VAR2 *)
            ST    VAR2
            LD    300               (* Default value for VAR3 *)
            ST    VAR3
            ...
            LD    TRUE
            S     EEPROM_WRITE      (* Force dump of V memory to EEPROM *)
            S     EEPROM_RESTORED   (* Activate alarm *)

MAIN_PROG:  LD    TRUE              (* Start of main program *)
            ...
            END

END_PROGRAM
```

The bit EEPROM_RESTORED (%M0.0) in this example will be activated to alert set an alarm in a higher level system that will tell the operator that memory has been corrupted, and default values have been restored. The operator should check if the values are correct or must be changed.

In some cases, the main program should not begin executing until correct parameters have been manually set by the operator. This could be accomplished in this way:

```
VAR_GLOBAL

      (* System bits *)
      EEPROM_ERROR AT %S0.7: BOOL;
      EEPROM_RESTORED AT %M0.0: BOOL;

END_VAR

PROGRAM Main

            LDN   EEPROM_ERROR
            OR    EEPROM_RESTORED
            JMPC  MAIN_PROG
            END

MAIN_PROG:  LD    TRUE              (* Start of main program *)
            ...
            END

END_PROGRAM
```

This program would keep waiting for the modbus master to restore the correct parameters. After that, the master should activate the EEPROM_RESTORED (%M0.0) bit, and then the main program will begin execution.

7.6.6.- Use of indirect addressing registers

The following program copies the first 50 registers from M memory to V memory, using indirect addressing registers and the LOOP instruction.

```
VAR_GLOBAL

      POINTER_M AT %SW40: WORD;
      POINTER_v AT %SW45: WORD;
      IND_M AT %SW50: WORD;
      IND_v AT %SW55: WORD;

END_VAR

PROGRAM Main

            LD    0           (* ACC = 0 *)
            ST    POINTER_M   (* Set pointer to %MW0 *)
            ST    POINTER_V   (* Set pointer to %VW0 *)

COPY_LOOP:  LD    IND_M       (* ACC = %MW[POINTER_M] *)
            ST    IND_V       (* %VW[POINTER_V] = ACC *)
            INC   POINTER_M   (* Increment pointer *)
            INC   POINTER_V   (* Increment pointer *)
            LD    POINTER_M
            LE    50          (* If pointer below 50... *)
            JMPC  COPY_LOOP   (* ...then repeat loop *)

MAIN_PROG:  LD    TRUE        (* Start of main program *)
            ...
            END

END_PROGRAM
```

8.- MODBUS COMMUNICATION

8.1.- Modbus protocol

Modbus is a master-slave protocol which normally operates over a RS-485 bus. In the bus there is only one device (the master) which initiates the modbus transactions, in order to read or write data to the other devices (the slaves). The slave devices reply to the master with the requested data, or with acknowledge or error responses.

This protocol has been chosed between many others because it's widespread use in the industrial area, and because it's a well-known, easy to use, well documented and free of use protocol. The modbus specifications are public and can be downloaded from www.modbus.org.

Communication frames are limited to 256 bytes, as the specification recommends. This allows reading or writing of about 100 16-bit registers with a single transaction.

Serial communications must be configured with 8 data bits, 1 stop bit and no parity. Baud rate must be 4800, 9600, 19200, 2880, 38400, 57600, 115200 or 128000 bps.

In the event of errors in the modbus requests, the SiAM CRIO board normally does not reply anything, but in certain cases it can reply with exception responses.

The board supports broadcast frames with destination address 0. These requests will never have a response.

8.2.- Modbus mapping

The SiAM CRIO board maps its inputs, outputs and memory registers to externally accessible modbus registers, according to the following mapping:

Modbus registers	Description	Access type
0x0000-0x00FF	Input memory (%IW0…%IW255)	Read-only
0x0100-0x01FF	Output memory (%QW0…%QW255)	Read/Write
0x0200-0x02FF	System memory (%SW0…%SW255)	Read/Write
0x0300-0x03FF	Reserved area	Read/Write
0x1000-0x1FFF	Modbus indirect addressing registers	Read/Write
0x2000-0x2FFF	Modbus indirect addressing configuration registers	Read/Write
0x4000-0x4FFF	RAM user memory (%MW0…%MW255)	Read/Write
0x5000-0x5FFF	EEPROM user memory (%VW0…%VW255)	Read/Write
0x6000-0x6FFF	System memory (%SW0…%SW255)	Read/Write
0x7000-0x7FFF	Configuration	Read-only
0x8000-0xFFFF	User program	Read-only

Only some of the 65536 available registers are used. The others are read-only registers that always return the value 0.

The **configuration** (0x7000-0x7FFF) and **user program** (0x8000-0xFFFF) memory areas are read-only areas. The configuration software SiAM CRIOSoft can also write data to these areas.

System memory is mapped to two different register areas (0x0200-0x02FF and 0x6000-0x6FFFF). Input, output, system, RAM and EEPROM memory areas have been described in detail in section 7.4. Indirect addressing areas –described also previously in section 6.8.2- and configuration area are described in the following points.

8.2.1.- Indirect addressing registers

Register	Description
0x1000…0x103F	Persistent indirect addressing registers. These virtual registers are redirected to the physical registers specified in 0x2000…0x201F as explained previously. So, if the register 0x2004 contains the value 10, accessing register 0x1004 will be the same that accessing register 10 (0x000A).
0x1100…0x113F	Volatile indirect addressing registers. These virtual registers are redirected to the physical registers specified in 0x2100…0x211F as explained previously. So, if the register 0x2104 contains the value 10, accessing register 0x1104 will be the same that accessing register 10 (0x000A).
0x2000…0x203F	Persistent indirect addressing configuration registers. These registers contain the number of register where the virtual registers 0x1000…0x101F are mapped. They are stored in non-volatile memory, so they keep their contents after power-down.
0x2100…0x213F	Volatile indirect addressing configuration registers. These registers contain the number of register where the virtual registers 0x1100…0x111F are mapped. They are stored in volatile memory, so they loose their contents after power-down (on power-up, all virtual registers 0x1000…0x111F point to physical register 0x0000).

8.2.2.- Configuration registers

Register	Description
0x7000	Device identifier. For this model, its value is always 7001 (0x1B59)
0x7001	Hardware version
0x7002	Firmware verions
0x7003…0x700F	Device identifier string. These registers contain the ASCII characters that idendity the device in the modbus network for the SiAM CRIOSoft software.
0x7010…0x7019	Reserved.
0x701A…0x7023	Lower limit of scaling for analog inputs AI0…AI7 (read/write). See section 6.8.3.
0x7024…0x702D	Highet limit of scaling for analog inputs AI0…AI7 (read/write). See section 6.8.3.
0x702E	Modbus link timeout in milliseconds (read/write). See section 6.8.1.

8.3.- Modbus functions

The SiAM CRIO borrad supports the modbus RTU protocol functions described in the following sections. As frames are limited to 256 bytes, is not possible to read or write more than a hundred modbus registers in a single transaction. In all examples the slave address is 128 (0x80).

8.3.1.- Read Holding Registers (0x03)

This function code is used to read the contents of a contiguous block of holding registers. The following example reads holding register 0x0000:

Request		Response	
Field name	Value (Hex)	Field name	Value (Hex.)
Slave address	0x80	Slave address	0x80
Function	0x03	Function	0x03
Starting address (HI)	0x00	Byte count	0x02
Starting address (LO)	0x00	Value of register 0x0000 (HI)	0x00
Number of registers (HI)	0x00	Value of register 0x0000 (LO)	0x00
Number of registers (LO)	0x01	Checksum (HI)	0x84
Checksum (HI)	0x9A	Checksum (LO)	0x5A
Checksum (LO)	0x1B		

8.3.2.- Read Input Registers (0x04)

Identical to the previous one. In this example, registers 0x0200-0x0201 are read:

Request		Response	
Field name	Value (Hex.)	Field name	Value (Hex.)
Slave address	0x80	Slave address	0x80
Function	0x04	Function	0x04
Starting address (HI)	0x02	Byte count	0x04
Starting address (LO)	0x00	Value of register 0x0200 (HI)	0x45
Number of registers (HI)	0x00	Value of register 0x0200 (LO)	0x02
Number of registers (LO)	0x02	Value of register 0x0201 (HI)	0x00
Checksum (HI)	0x6E	Value of register 0x0201 (LO)	0x00
Checksum (LO)	0x62	Checksum (HI)	0xDE
		Checksum (LO)	0x40

The read value of register 0x0200 is 0x4502, and the value of register 0x0201 is 0x0000.

8.3.3.- Write Single Register (0x06)

This function allows writing of a single holding register. This example writes the value 0xFFFF in register 0x0100:

Request		Response	
Field name	Value (Hex.)	Field name	Value (Hex.)
Slave address	0x80	Slave address	0x80
Function	0x06	Function	0x06
Register address (HI)	0x01	Register address (HI)	0x01
Register address (LO)	0x00	Register address (LO)	0x00
Register value (HI)	0xFF	Register value (HI)	0xFF
Register value (LO)	0xFF	Register value (LO)	0xFF
Checksum (HI)	0x9A	Checksum (HI)	0x9A
Checksum (LO)	0x1B	Checksum (LO)	0x1B

8.3.4.- Write Multple Registers (0x10)

This function code is used to write a block of contiguous registers. This example, as the previous one, writes the value 0xFFFF in register 0x0100, but using this function instead:

Request		Response	
Field name	Value (Hex.)	Field name	Value (Hex.)
Slave address	0x80	Slave address	0x80
Function	0x10	Function	0x10
Starting address (HI)	0x01	Starting address (HI)	0x01
Starting address (LO)	0x00	Starting address (LO)	0x00
Quantity of registers (HI)	0x00	Quantity of registers (HI)	0x00
Quantity of registers (LO)	0x01	Quantity of registers (LO)	0x01
Byte count	0x02	Checksum (HI)	0x1E
Value of register 0x0100 (HI)	0xFF	Checksum (LO)	0x24
Value of register 0x0100 (LO)	0xFF		
Checksum (HI)	0xDB		
Checksum (LO)	0x76		

8.3.5.- Mask Write Register (0x16)

This function code is used to modify the contents of a specified holding register using a combination of an AND mask, an OR mask, and the register's current contents. The function can be used to set or clear individual bits in the register, in this way:

```
Result = (Current_Contents AND AND_MASK) OR (OR_MASK AND (NOT AND_MASK))
```

So, if the OR_MASK value is zero, the result is simply the logical AND of the current contents and AND_MASK. If the AND_MASK value is zero, the result is equal to the OR_MASK value.

This example activates bits 5 and 6 in register 0x0300:

Request		Response	
Field name	Value (Hex.)	Field name	Value (Hex.)
Slave address	0x80	Slave address	0x80
Function	0x16	Function	0x16
Register address (HI)	0x03	Register address (HI)	0x03
Register address (LO)	0x00	Register address (LO)	0x00
AND mask (HI)	0x00	AND mask (HI)	0x00
AND mask (LO)	0x00	AND mask (LO)	0x00
OR mask (HI)	0x00	OR mask (HI)	0x00
OR mask (LO)	0x60	OR mask (LO)	0x60
Checksum (HI)	0x3F	Checksum (HI)	0x3F
Checksum (LO)	0xB1	Checksum (LO)	0xB1

8.3.6.- Read/Write Multiple Registers (0x17)

This function code performs a combination of one read operation and one write operation in a single modbus transaction. The write operation is performed before the read. It's very useful to read inputs and write outputs very quickly, as shown in this example:

Request		Response	
Field name	Value (Hex.)	Field name	Value (Hex.)
Slave address	0x80	Slave address	0x80
Function	0x17	Function	0x17
Read starting address (HI)	0x00	Byte count	0x02
Read starting address (LO)	0x00	Value of register 0x0000 (HI)	0x00
Quantity to read (HI)	0x00	Value of register 0x0000 (LO)	0x00
Quantity to read (LO)	0x01	Checksum (HI)	0x81
Write starting address (HI)	0x01	Checksum (LO)	0xAA
Write starting address (LO)	0x00		
Quantity to write (HI)	0x00		
Quantity to write (LO)	0x01		
Byte count	0x02		
Value of register 0x0100 (HI)	0xFF		
Value of register 0x0100 (LO)	0xFF		
Checksum (HI)	0xB8		
Checksum (LO)	0x9E		

9.- FREQUENTLY ASKED QUESTIONS

9.1.- Communication with the board can not be established

If communication can't be established, check the following points:

1. Check RS-485 bus wiring in SiAM CRIO board and in all the other devices in the bus (other boards, RS232/485 converters, etc).
2. Check that configuration jumpers in the board are correctly adjusted:
    a. Bus termination resistor must be active only in the first and last devices in the RS-485 bus.
    b. Pull-up and pull-down resistors must be active in only one device in the bus.
3. If still there's no communication, watch communication diagnostics LEDs:
    a. If RX LED is not lit, then the board is not receiving data from the bus. There can be a problem with wiring, or the modbus master is not sending data (maybe it's stopped or not properly connected).
    b. If the RX LED lits but the TX LED doesn't, then the SiAM CRIO board is not responding to the modbus requests. This can happen if the requests are not addressed to the correct modbus slave. Check the slave address the master is trying to communicate to.
4. If all is correct, then the communication problem can be due to a hardware problem. If the main state LED is not blinking, then the microcontroller can be damaged. If it blinks but communication is still not possible, then maybe some communication component in the board has stopped functioning, maybe because some overvoltage in the bus.

9.2.- Digital outputs are all inactive

Check the online status of the outputs with the monitoring software. If the outputs are not active in the software, then make sure that the program is activating them. If they are but their corresponding LEDs are not active, check the outputs power supply terminals to see if you're supplying power to them (9-30VDC). See section 4.3 for a detatiled description of of digital output connectors.

9.3.- The IEC code is not executing correctly

If the instruction are not executed in an unexpected way, it's probably because the logic accumulator RLO has the value FALSE. This disables execution of some instructions (for example, all the conditional instructions which end in 'C' like JMPC, CALC, RETC…). To avoid this, after finishing a logical segment of the program (typically with a store instruction like ST or STN), place an instruction to start the new logical segment with the RLO value to TRUE. This can be done with LD TRUE, or better with an S instruction (SET without operand) which consumes less program memory.

10.- ELECTRICAL CHARACTERISTICS

Parameter	Min.	Typ.	Max.	Units
Temperature range	0	-	70	ºC
Supply				
Main supply voltage V_{IN}	9	24	30	V
Current consumtion	15[10]	-	110[11]	mA
Power consumption - V_{IN} = 9 VDC	0,1	-	1	W
Power consumption - V_{IN} = 12 VDC	0,2	-	1,3	W
Power consumption - V_{IN} = 24 VDC	0,4	-	2,6	W
Digital inputs				
Input impedance	-	3,9	-	KΩ
Per input current consumption - V_{IN} = 9VDC	-	-	2,3	mA
Per input current consumption - V_{IN} = 12VDC	-	-	3,1	mA
Per input current consumption - V_{IN} = 24VDC	-	-	6,2	mA
Analog inputs				
Input impedance – Current mode (0-20mA)	-	250	-	Ω
Digital outputs				
Outputs supply voltage $V_{OUTPUTS}$	9	24	30	V
Maximum output current	-	-	80	mA
Program				
Program memory	8192 bytes (> 1300 IEC instructions)			
Non-retentive (RAM) user memory	256 words			
Retentive (EEPROM) user memory	256 words			
EEPROM durability	100K	1M	-	Writes
Maximum paréntesis nesting levels	4			
Maximum subroutine nesting levels	4			
IEC instruction execution time[12]	-	75	150	µs

[10] No outputs active
[11] All outputs active
[12] Depends on type of instruction and can increase if program online monitoring is active

11.- COPYRIGHT

This product with all its components and this manual are property of Software i Automatismes Morvedre S.L. Total or partial reproduction of design, program, electronic board or manual is forbidden. Any action against this statement will be prosecuted according to intellectual property laws.

April 2009

Software i Automatismes Morvedre, S. L.
Vent de Ponent, 22
46520 Puerto de Sagunto
Valencia (Spain)
www.siam.es
info@siam.es